PRIME Technical Update

Subject: PRIMOS FILE SYSTEM, REV. 13

Number: 30

Revision: 0

Date: May 1976

Applicable Hardware: All CPU's

Applicable Software: PRIMOS III and IV, Rev. 13

Documentation Impact: Supplements MAN2604

Abstract:

A number of new features have been introduced into the file system for REV. 13 of PRIMOS III and PRIMOS IV. Among these changes are 32-character filenames, fully indexed DAM files, multi-record UFDs, and a new set of file system subroutines that support the new capabilities.

The new file system facilities are introduced on a partition basis, that is, a disk partition is either "old" or "new", but not a mixture of old and new files. All existing programs that use the file system will continue to run on old partitions and, with certain restrictions noted herein, on new partitions.

Part 1 of this document details the new file system features, describes the new file system CALLs, and gives program examples of how the new file system is used.

Part 2 describes the new functions of the FUTIL file utility.

This bulletin is one in a series of documentation supplements that supply current information on Prime hardware, software and documentation products. Prime Technical Updates introduce product improvements and revisions, and update existing Prime Computer user documentation.

PRIME Computer, Inc. 145 Pennsylvania Avenue, Framingham, Mass. 01701/(617) 879-2960

P/N PTU-3065

CONTENTS

Part 1 Rev. 13 File System Changes

CONTENTS

Part 2 FUTIL, Rev. 12 and 13

PART 1

REV. 13 FILE SYSTEM CHANGES

1.1 INTRODUCTION

1.1.1 MOTIVATION

The REV. 13 file system represents the first step in an evolutionary process in which the capabilities of the current file system are to be greatly upgraded and expanded. The new capabilities at REV. 13 include 32-character filenames, longer files and UFDs, date/time stamping, new error handling, and more secure handling of segment directories and UFDs. The design of the new file system has been guided by three principles:

1) Whenever possible, existing user programs should continue to work on the new file system without modifications.

2) The internal formats and functionality of the new file system should allow future expansion without affecting programs using the new file system. Program efficiency should not be penalized by the introduction of rarely used features.

3) Requirements for accessing and modifying existing file structures are more stringent. Access rights are carefully observed, and UFD and segment directory modification is more controlled.

1.1.2 COMPATIBILITY OF OLD AND NEW PARTITIONS

It should be stressed that the distinction between old and new file system features is on a partition basis. A partition is either in the old format or in the new format, never a mixture of the two. The following comments apply to the interaction of old/new CALLs on old/new partitions.

OLD CALLS ON OLD PARTITIONS: Existing programs and new programs that use the old file system calls will continue to work without modification on old partitions.

OLD CALLS ON NEW PARTITIONS: Existing programs will continue to work on new partitions with the following exceptions: PRWFIL can no longer be used in any way on directories (UFDs or segment directories). SEARCH can not be used to delete a non-empty directory. Segment directories can no longer have UFD subentries. There are certain new restrictions on filenames in new partitions (see Section 1.2.1.1).

NEW CALLS ON OLD PARTITIONS: Programs using the new file system calls will work correctly on old partitions. Certain error cases will, of course, arise when trying to perform a function only supported by a new partition Any filename specified as longer than six characters is truncated to six characters when running on an old partition.

NEW CALLS ON NEW PARTITIONS: All functions described in this document will work on new partitions.

Note also that using the new file system, it is still possible to run with write-protected disks.

1.1.3 PRIMOS SUPPORT FOR NEW FILE SYSTEM

At REV. 13, all utilities support the new file system. Certain commands cannot take a filename longer than six characters. These are AVAIL, BASINP, CNVIMA, CRPMC, CRSER, CX, EDB, FILVER, MCG, NUMBER, PRMPC, PRSER, PRVER, PUSS, TRAMLC, UPCASE.

1.1.4 PRIMOS II AND III CONSIDERATIONS

New file system calls new partitions are not supported by PRIMOS II. In addition, new file system calls are not supported by PRINET on calls to access remote disks. To support calls in these situations, a package -- the BOUNCE package -- is available that will "bounce" the new calls back to user space where they will be transformed into a series of old calls that are understood by PRIMOS II. In some cases (noted below), file unit 16 is used by the bounce package. In others, such as GPAS$$ and SATR$$, the current UFD is opened by the bounce package. The bounce package is described in Section 1.6.

1.2 OVERVIEW OF NEW FILE SYSTEM FEATURES

1.2.1 NEW FILE CHARACTERISTICS

1.2.1.1 New File Names

New filenames can be up to 32 characters long, the first character of which must not be numeric (0-9). Filenames can be composed only of the following characters:

A-Z 0-9 _ # $ & * - . /

If any lower case characters are specified, the are forced to upper case. No control characters (0 - :237) are allowed in new file names. In the new file system calls, file names are, as before, ASCII packed two characters per word. If the name length specified in a call is longer than the actual length of the name, the name must be followed by a number of trailing blanks sufficient to match the given length.

1.2.1.2 Date/Time Stamping

There is a new field in a file's UFD entry that records the date and time when the file was last modified. This field is updated when a file is closed and:

1) An old file has been opened for K$WRIT or K$RDWR and a write operation has been performed.

2) A new file has been created.

(Note: the decision to use "last modified" rather than "last used" was to allow the use of write-protected disks.)

1.2.1.3 Unlimited Partition and File Size

On old partitions, the number of records in a file and the number of records in a partition are limited to 65536. The number of records in a new file or partition is now effectively unlimited and can fill any physical storage device supported by PRIMOS. A storage module disk partition containing more than 8 heads must be a new partition.

1.2.1.4 Fully Indexed DAM Files

Formerly, when the index of a DAM file overflowed one record (1024 entries for a storage module, 440 entries for all other disks), access became sequential. In new DAM files, a multi-level index is maintained so that any record in the file can be directly accessed. With the exception of improved access time, this difference is invisible to user programs. (More details on the new DAM organization are given in Section 1.8.5.)

1.2.2 NEW UFD CHARACTERISTICS

1.2.2.1 Multi-Record UFDs

UFDS, formerly restricted to a single record, can now span multiple records. The limit of no more than 72 files (169 on a storage module) in a single UFD no longer holds. (The UFD FULL message will never be generated.)

1.2.2.2 Hidden Internal Format

It is no longer possible to read and write a UFD using PRWFIL (or the new PRWF$$). Indeed, there is no need for a program to know the internal format of a UFD. Programs are therefore protected from future changes to the file system. The new way in which UFDs are read is detailed under the description of the RDEN$$ subroutine in Section 1.3.

1.2.2.3 Special File Identification

UFD entries now include an identification of "special" files -- files having unique use in the file system and not normally accessed by the user. These files are BOOT, DSKRAT, BADSPT, and MFD.

1.2.3 NEW SEGMENT DIRECTORIES

1.2.3.1 New Entry Identification

Entries in a segment directory are no longer identified by a <record-number, word-number> pair, but by a single entry number from 0 to 65535. This means that segment directories are now limited in size to 65536 entries (0 - :177777).

1.2.3.2 Segment Directory Handling

It is no longer possible to read and write segment directories using PRWFIL (or the new PRWF$$). A new subroutine -- SGDR$$ -- is provided for the examination and modification of segment directory entries.

1.2.3.3 Segment Directory Restriction

A UFD entry in a segment directory is now illegal. The only filetypes allowed in a segment directory are SAM, DAM, and other segment directories. This restriction applies to both new and old partitions.

1.3 NEW FILE SYSTEM SUBROUTINES

1.3.1 NOTES ON SUBROUTINE DESCRIPTIONS

For each subroutine a complete description of the parameters is given, followed by notes on usage, brief examples of calls, and notes on compatibility with old file system functions. Error return codes are summarized in a table following the subroutine descriptions.

Section 1.7 illustrates use of the subroutines with more complete sample programs. Throughout, it is assumed that the reader is familiar with the old file system capabilities as described, for example, in the PRIMOS III or IV User's Guide (MAN2604).

1.3.2 KEY DEFINITIONS FOR NEW FILE SYSTEM CALLS

All keys and error codes are specified in symbolic, rather than numeric, form. These symbolic names are defined as PARAMETERs (for FORTRAN programs) and EQUs (for PMA programs) in $INSERT files present in a new UFD on the master disk called SYSCOM. The key definition files are named KEYS.F for FORTRAN and KEYS.P for PMA. The error definition files are ERRD.F and ERRD.P. For convenience in recognizing old file system keys, a listing of these files are included in Section 1.4. The user is urged to use these symbolic names.

1.3.3 NEW ERROR HANDLING CONVENTIONS

All alternate return parameters (ALTRTN) have been replaced with CODE -- an integer return code variable. This is part of the new error handling protocol, which is completely described in Section 1.5.

1.3.4 SUBROUTINE DESCRIPTIONS

1.3.4.1 ATCH$$ -- Attach to UFD

Function

Attach to a UFD and optionally make it the home UFD.

Calling Sequence

CALL ATCH$$ (UFDNAM,NAMLEN,LDISK,PASSWD,KEY,CODE)

Parameters

UFDNAM — The name of the UFD to be attached to. If KEY=0 and UFDNAM is the key K$HOME the home UFD is attached.

NAMLEN — The length in characters of UFDNAM. NAMLEN may be greater than the actual length of UFDNAM if UFDNAM is padded with the appropriate number of blanks. If UFDNAM=K$HOME, NAMLEN is disregarded.

LDISK — The number of the logical disk to be searched for UFDNAM when KEY=K$IMFD. Other values are:

K$ALLD--Search all started-up logical devices.

K$CURR--Search the MFD of the disk currently attached.

PASSWD — A three-word array containing one of the passwords of UFDNAM. Can be specified as 0 if attaching to the home UFD.

KEY — A reference value as follows:

K$IMFD--Attach to UFDNAM in MFD on LDISK.

K$ICUR--Attach to UFDNAM in current UFD (UFDNAM is a subdirectory).

To these two keys may be added K$SETH, e.g., K$IMFD+K$SETH, which will set the current UFD to the home UFD after attaching.

CODE An integer variable set to the return code.

Notes on Usage

A BAD PASSWD error is not returned to the user's program. Command level is entered, and the user is left attached to no UFD. Other errors leave the attach point unchanged.

Examples

1) Attach to home UFD:

```
CALL ATCH$$ (K$HOME,0,0,0,0,CODE)
```

2) Attach to UFD named 'G.S.PATTON', password 'CHARGE' in current UFD:

```
CALL ATCH$$ ('G.S.PATTON',10,K$CURR,'CHARGE',K$ICUR,CODE)
```

Compatibility

ATCH$$ is equivalent to ATTACH with the addition of support for longer names.

1.3.4.2 COMI$$ -- Switch Command Input Stream

Function

COMI$$ is used to switch the command input stream from the terminal to a command file, or from a command file to the terminal.

Calling Sequence

```
CALL COMI$$(NAME,NAMLEN,FUNIT,CODE)
```

Parameters

NAME — The name of the file to switch the command input stream. If NAME is 'TTY', the command stream is switched back to the terminal and FUNIT is closed. if NAME is 'PAUSE', the command stream is switched to the terminal but FUNIT is not closed. If NAME is 'CONINUE', the command stream is switched to the file already open on FUNIT.

NAMLEN — The length in characters of NAME.

FUNIT — The file unit on which to open the command file specified by NAME. Normally, file unit six is used.

CODE — An integer variable set to the return code.

Compatibility

COMI$$ has the same function as COMINP extended for long names. COMI$$ works on both new and old partitions. Note that neither COMINP nor COMI$$ currently works across the PRIMENET.

1.3.4.3 CREA$$ -- Create a New UFD in the Current UFD

Function

CREA$$ creates a new UFD (a SUBUFD) in the current UFD and initializes the new UFD entry. It replaces the former SEARCH NEWUFD subkey, used to create new UFDs on an old partition.

Calling Sequence

```
CALL CREA$$ (NAME,NAMLEN,OPASS,NPASS,CODE)
```

Parameters

NAME — The name to be given the new UFD.

NAMLEN — The length in characters of NAME.

OPASS — A three-word array containing the owner password for the new UFD. If OPASS(1)=Ø, the owner password is set to blanks.

NPASS — A three-word array containing the non-owner password for the new UFD. If NPASS(1)=Ø the non-owner password is set to Ø's. Any password given to ATTACH or ATCH$$ will match a non-owner password of Ø's.

CODE — An integer variable to be set to the return code from CREA$$.

Notes on Usage

Passwords can be at most 6 characters long. Passwords less than 6 characters must be padded with blanks for the remaining characters. Passwords are not restricted by filename conventions and may contain any characters or bit patterns. It is strongly recommended that passwords not contain blanks, commas, or the characters =,!,',@,{,},[,],(,) or lowercase characters. Passwords should not start with a digit. If passwords contain any of the above characters or begin with a digit, the passwords may not be given on a PRIMOS command line to the ATTACH command. line to the ATTACH command.

Since the new SEARCH, SRCH$$, will not allow creation of a new UFD, CREA$$ must be used for this purpose.

CREA$$ requires owner-rights on the current UFD.

If the bounce package is invoked (see Section 6), file unit 16 is used during the create. Unit 16 should not be open when CREA$$ is called.

Examples

1) Create new UFD with default passwords of ' ' for owner and 3*0 for non-owner:

```
CALL CREA$$ ('NEWUFD',6,0,0,CODE)
```

Compatibility

CREA$$ has no corresponding old file system subroutine. CREAT$ works on both old and new partitions.

1.3.4.4 CNAM$$ -- Change a Filename

Function

CNAM$$ is used to change the name of a file in the current ufd.

Calling Sequence

```
CALL CNAM$$(OLDNAM,OLDLEN,NEWNAM,NEWLEN,CODE)
```

Parameters

OLDNAM	The name of the file to be changed.
OLDLEN	The length in characters of OLDNAM.
NEWNAM	The name to be changed to.
NEWLEN	The length in characters of NEWLEN.
CODE	An integer variable set to the return code.

Notes on Usage

The user must be the owner to change the name. CNAM$$ does not change the last modified date-time of the file or any of the other attributes of the file.However, the last modified date-time of the UFD in which the file resides is changed. On a new partition, CNAM$$ may cause the position of the file in the UFD to change with respect to the other files. It is illegal to change the name of the MFD, BOOT, BADSPT, or the packname. A NO RIGHT error message is generated if this is attempted.

Compatibility

CNAM$$ provides the functionality of CNAME$ extended for long names. CNAM$$ is not available under PRIMOS II or across the PRIMENET.

1.3.4.5 GPAS$$ -- Obtain UFD Passwords

Function

GPAS$$ returns the passwords of a SUBUFD in the current UFD.

Calling Sequence

CALL GPAS$$ (UFDNAM,NAMLEN,OPASS,NPASS,CODE)

Parameters

UFDNAM The name of the UFD whose passwords are to be returned. UFDNAM is searched for in the current UFD.

NAMLEN The length in characters of UFDNAM.

OPASS A three-word array that is set to the owner password of UFDNAM.

NPASS A three-word array that is set to the non-owner password of UFDNAM.

CODE An integer variable set to the return code.

Notes on Usage

On the old file system it was possible to obtain the passwords of a UFD by reading the UFD's header with PRWFIL. On new partitions it is not possible to read a UFD with PRWFIL or PRWF$$ -- GPAS$$ must be used.

GPAS$$ requires owner-rights to the current UFD.

If the bounce package is invoked (see Section 1.6) file unit 16 is used, and the current UFD is opened for reading, then closed. Therefore, when GPAS$$ is called unit 16 should be closed, and the current UFD should not be open for writing on any unit.

Examples

1) Read passwords of SUBUFD into PASS(6) array:

```
CALL GPAS$$ ('SUBUFD',6,PASS(1),PASS(4),CODE)
```

Compatibility

GPAS$$ corresponds to no old file system subroutine. GPAS$$ works on both old and new partitions.

1.3.4.6 NAMEQ$ -- Compare Filenames

Function

NAMEQ$ is a LOGICAL function that compares two filenames for equivalence.

Calling Sequence

<logical> = NAMEQ$ (NAME1,LEN1,NAME2,LEN2)

Parameters

NAME1	The first filename for comparison.
LEN1	The length in characters of NAME1.
NAME2	The second filename for comparison.
LEN2	The length in characters of NAME2.

Notes on Usage

NAMEQ$ does a character-by-character compare of NAME1 and NAME2 up to LEN1 or LEN2, whichever is shorter. The trailing characters of the longer name (if the names are not the same length) must all be blank for equality.

NAMEQ$ will work correctly on numeric fields only if LEN1=LEN2.

Examples

1) The following sets EQUAL to .TRUE. no matter what is in ARRAY:

```
EQUAL=NAMEQ$(ARRAY(1),127,ARRAY(1),127)
```

2) FNAME(3) must be ' ' for the following to set EQUAL .TRUE.:

```
EQUAL = NAMEQ$ (FNAME(1),6,'NAME',4)
```

Compatibility

NAMEQ$ provides the functionality of NAMEQV extended for varying length character strings.

1.3.4.7 PRWF$$ -- Read-Write-Position SAM or DAM File

Function

PRWF$$ is used to read, write, position, and truncate SAM or DAM files.

Calling Sequence

```
CALL PRWF$$ (RWKEY+POSKEY+MODE,FUNIT,LOC(BUFFER),NW,
                                                POS,RNW,CODE)
```

Parameters

RWKEY This subkey, which cannot be omitted, indicates the action to be taken. Possible values are:

K$READ--Read NW words from FUNIT into BUFFER.

K$WRIT--Write NW words from BUFFER to FUNIT.

K$POSN--Set the current position to the 32-bit integer in POS.

K$TRNC--Truncate the file open on FUNIT at the current position.

K$RPOS--Return the current position as a 32-bit integer word number in POS.

POSKEY A subkey indicating the positioning to be performed. Possible values are:

K$PRER--Move the file pointer of FUNIT POS words relative to the current position before performing RWKEY.

K$POSR--Move the file pointer of FUNIT POS words relative to the current position after performing RWKEY.

K$PREA--Move the file pointer of FUNIT to the absolute position specified by POS before performing RWKEY.

K$POSA--Move the file pointer of FUNIT to the absolute position specified by POS after performing RWKEY.

Note: if this subkey is omitted, the default action is that of K$PRER.

MODE A subkey that is either omitted or has the value K$CONV. If omitted, NW words are read or written. If not omitted, a convenient number of words (up to NW) is read or written. (The meaning of "convenient" is described in the PRIMOS User's Guide.)

FUNIT A file unit number from 1 to 16 (1 to 15 for PRIMOS II or under PRINET) on which a file has been opened by a call to SRCH$$ or by a command. PRWF$$ actions are performed on this file unit.

BUFFER The data buffer to be used for reading or writing. If BUFFER is not needed, it can be specified as LOC(0).

NW The number of words to be read or written (MODE=0) or the maximum number of words to be transferred (MODE=K$CONV). NW may be between 0 and 65535.

POS A 32-bit integer (INTEGER*4) specifying the relative or absolute positioning value depending on the value of POSKEY.

RNW A 16-bit unsigned integer set to the number of words actually transferred when RWKEY=K$READ or K$WRIT. Other keys leave RNW unmodified. For the keys K$READ and K$WRIT, RNW must be specified.

CODE An integer variable to be set to the return code.

Notes on Usage

POS is always a 32-bit integer, not a <record-number, word-number> pair. All calls to PRWF$$ must specify POS even if no positioning is requested. An INTEGER*4 0 can be generated by specifying "000000" or "INTL(0)" in FTN, "0L" in PMA.

POSKEY is observed for all values of RWKEY except REDPOS, for which it is ignored (the file position is never changed).

If RWKEY = K$POSN, NW and RNW are ignored, and no data is transferred.

Note that it is no longer necessary to call GETERR to obtain the number of words transferred.

Examples

1) Read the next 79 words from the file open on unit 1:

```
CALL PRWF$$ (K$READ,1,LOC(BUFFER),79,000000,NMREAD,CODE)
```

2) Add 1024 words to the end of the file open on UNIT (10000000 is just a very large number to get to the end of the file):

```
CALL PRWF$$ (K$POSN+K$PREA,UNIT,LOC(0),0,10000000,NMW,CODE)
CALL PRWF$$ (K$WRIT,UNIT,LOC(BFR),1024,000000,NMW,CODE)
```

3) See what position is on file unit 15 (INT4 is INTEGER*4):

```
CALL PRWF$$ (REDPOS,15,LOC(0),0,INT4,0,CODE)
```

4) Truncate file 10 words beyond the position returned by the above call:

```
CALL PRWF$$ (K$TRNC+K$PREA,15,LOC(0),0,INT4+10,0,CODE)
```

Compatibility

PRWF$$ cannot be used on UFDs or segment directories as could PRWFIL. Note that PRWF$$ now performs the TRUNCATE function, formerly associated with SEARCH. The REWIND function of SEARCH is also performed by PRWF$$: to rewind a file perform the following call:

```
CALL PRWF$$ (K$POSN+K$PREA,FUNIT,0,0,000000,RNW,CODE)
```

This will position to the start of the file without performing any data transfer.

1.3.4.8 RDEN$$ -- Read UFD Entry

Function

RDEN$$ positions-in or reads-from a UFD.

Calling Sequence

CALL RDEN$$ (KEY,FUNIT,BUFFER,BUFLEN,RNW,NAME,NAMLEN,CODE)

Parameters

KEY — An integer variable specifying the action to be taken. Possible values are:

K$READ--Advance to the start of the first or next UFD entry and read as much of the entry as will fit into BUFFER. Set RNW to the number of words read.

K$GPOS--Return the current position in the UFD as a 32-bit integer in NAME.

K$UPOS--Set the current position in the UFD from the 32-bit integer in NAME.

FUNIT — A unit on which a UFD is currently opened for reading. (A UFD may be opened with a call to SRCH$$.)

BUFFER — A one dimensional array into which entries of the UFD are read.

BUFLEN — The length in words of BUFFER.

RNW — An integer variable that will be set to the number of words read.

NAME — A 32-bit integer variable used for keys of GETPOS and SETPOS.

NAMLEN — A 16-bit integer variable reserved for future use. (It is envisioned that NAME and NAMLEN will in the future be used to allow searching for the entry corresponding to a particular filename.)

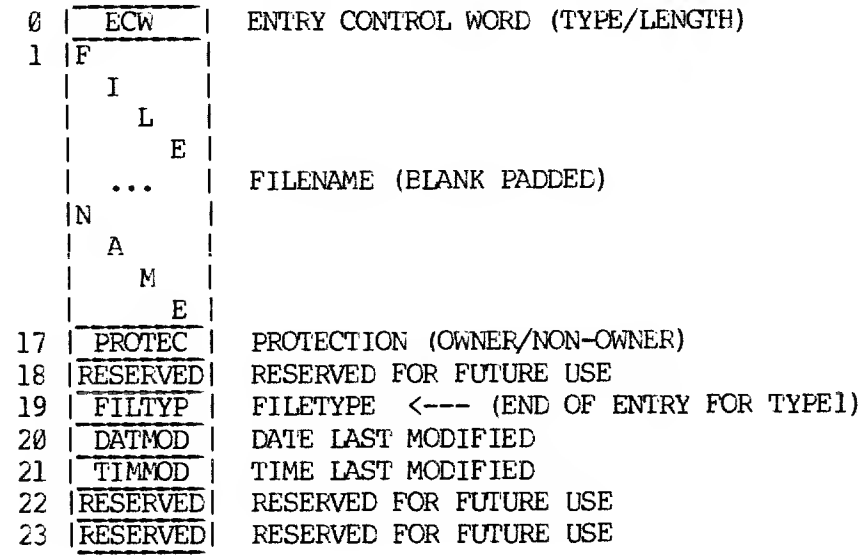
CODE — An integer variable to be set to the return code.

Notes on Usage

RDEN$$ is used to read entries from a UFD. RNW words are returned in BUFFER, and the file unit position is advanced to the start of the next entry. Return code E$EOF means no more entries, E$BFTS means BUFFER is too small for the entry.

Note that in the new file system, UFDs are not compressed when files are deleted, and vacant entries may be reused. Thus, a newly-created file will not necessarily be found at the end of a UFD.

The complete format of currently defined entries is given here. (All numbers are decimal unless preceded by a ':'.)



ECW Entry Control Word. An ECW is the first word in any entry and consists of two 8-bit subfields. The high-order 8 bits indicate the type of the entry, the low-order 8 bits give the length of the entry in words including the ECW itself. Possible values of the ECW at REV. 13 are as follows:

:000001 - Type=0, length=1. This entry indicates either a UFD header or a vacant entry. No information other than the ECW is returned.

:000424 - Type=1, length=20. Type=1 indicates an old UFD entry. Words 0-19 in the diagram above are returned.

:001030 - Type=2, length=24. Type=2 indicates a new UFD entry. All the above information is returned. Reserved fields should be ignored.

User programs should ignore any entry-types that are not recognized. This will allow future expansion of the file system without unduly affecting old programs.

FILENAME Up to 32 characters of filename, blank padded.

PROTEC Owner and non-owner protection attributes. The owner rights are in the high-order 8 bits, the non-owner in the low-order 8 bits. The meanings of the bit positions are as follows (a 1-bit grants the indicated access right):

1-5,9-13	Reserved for future use.
6,14	Delete/truncate rights.
7,15	Write-access rights.
8,16	Read-access rights.

FILTYP On a new partition, the low-order 8 bits indicate the type of the file as follows:

0	SAM file.
1	DAM file.
2	SAM Segment directory.
3	DAM Segment Directory.
4	UFD

On an old partition, the filetype is zero -- the file must be opened with SRCH$$ to determine its type. Of the high-order 8 bits, only bit 4 (:10000) is currently defined. If one, it indicates a special file -- BOOT, MFD, DSKRAT, or BADSPT. The other bits are reserved for future use. (Bit 4 is valid on both new and old partitions.)

DATMOD The date on which the file was last modified. The date, which is valid only on new partitions, is held in the binary form YYYYYYYMMMMDDDDD, where YYYYYYY is the year modulo 100, MMMM is the month, DDDDD is the day.

TIMMOD The time at which the file was last modified. The time, which is valid only on new partitions, is held in binary seconds-since-midnight divided by four.

Examples

1) Read next entry from new or old UFD:

```
100     CALL RDEN$$ (K$READ,FUNIT,ENTRY,24,RNW,0,0,CODE)
        IF (CODE .NE. 0) GOTO <error handler>
        TYPE=RS(ENTRY(1),8)  /* GET TYPE OF ENTRY JUST READ
        IF (TYPE.NE.1.AND.TYPE.NE.2) GOTO 100 /* UNKNOWN
```

2) Position to beginning of UFD:

```
        CALL RDEN$$ (K$UPOS,FUNIT,0,0,0,000000,0,CODE)
```

Compatibility

RDEN$$ provides the facility lacking in the new PRWF$$ -- the ability to read UFDs. In addition, knowledge of the internal layout of a UFD is not necessary in user programs. RDEN$$ can be used on both old and new partition UFDs.

1.3.4.9 REST$$ -- Restor a P300 Memory Image from a File

Function

REST$$ reads a P300 memory image from a file in the current UFD into memory. The SAVE'd parameters for a file previosly written to the disk by the SAVE or SAVE$$ subroutine or the SAVE command are loaded into the nine word array VECTOR. The memory image itself is then loaded into memory using the starting and ending addressed provided by VECTOR(1) and VECTOR(2).

Calling Sequence

```
CALL REST$$(VECTOR,NAME,NAMLEN,CODE)
```

Parameters

VECTOR — A nine word array set by REST$$. VECTOR(1) is set to the first location in memory to be restored. VECTOR(2) is set to the last location to be restored. The rest of the array is set as follows:

```
VECTOR(3) saved P register
VECTOR(4) saved A register
VECTOR(5) saved B register
VECTOR(6) saved X register
VECTOR(7) saved Keys
VECTOR(8) not used
VECTOR(9) not used
```

NAME — The name of the file containing the memory image.

NAMLEN — The length in characters of NAME.

CODE — An integer variable set to the return code.

Compatibility

REST$$ has the same function as RESTOR and handles long names. REST$$ works on both old and new partitions.

1.3.4.10 RESU$$ -- Resuming a P300 Memory Image File

Function

RESU$$ restores a P300 memory image from a file in the current UFD, initializes registers from the saved parameters, and starts executing the program.

Calling Sequence

```
CALL RESU$$(NAME,NAMLEN)
```

Parameters

NAME The name of the file containing the memory image.

NAMLEN The length in characters of NAME.

Notes on Usage

RESU$$ does not have a CODE argument. On a error, an error message is typed and control returns to command level.

Compatibility

RESU$$ provides the functionality of RESUME extended for long names. RESU$$ works both on old and new partitions.

1.3.4.11 SATR$$ -- Set Attributes in UFD Entry

Function

SATR$$ allows the setting or modification of a file's attributes in its UFD entry.

Calling Sequence

```
CALL SATR$$ (KEY,NAME,NAMLEN,ARRAY,CODE)
```

Parameters

KEY — An integer variable specifying the action to take. Possible values are:

K$PROT--Set protection attributes from ARRAY(1). ARRAY(2) is ignored for old partitions and must be Ø for new partitions (it is reserved for expansion). The meaning of the protection bits in ARRAY(1) is given under RDEN$$ above.

K$DTIM--Set date/time modified from ARRAY(1) and ARRAY(2). The format of the date/time is given under RDEN$$ above.

NAME — The name of the file whose attributes are to be modified. The current UFD is searched for NAME.

NAMLEN — The length in characters of NAME.

ARRAY — A two-word array containing the attributes. For K$PROT, ARRAY(2) must be zero.

CODE — An integer variable set to the return code.

Notes on Usage

Owner rights are required on the UFD containing the entry to be modified.

The formats of the attributes in ARRAY are the same as those in a UFD entry obtained from RDEN$$.

An attempt to set the date/time modified on an old partition will result in an E$OLDP error (error message 'OLD PARTITION').

Since a call to SATR$$ modifies the UFD, the date/time modified of the UFD itself is updated.

If the bounce package is being used (see Section 1.6), file unit 16 should be closed, and the current UFD should not be open on any unit prior to the call.

Examples

1) Set default protection attributes on MYFILE:

```
ARRAY(1)=:3400 /* OWNER=7, NON-OWNER=0
ARRAY(2)=0     /* SECOND WORD MUST BE 0
CALL SATR$$ (K$PROT,'MYFILE',6,ARRAY(1),CODE)
```

2) Set both owner and non-owner attributes to read-only (note carefully bit positioning in two-word octal constant):

```
CALL SATR$$ (K$PROT,'NO-YOU-DON''T',12,:100200000,CODE)
```

3) Set date/time modified from UFD entry read into ENTRY by RDEN$$:

```
CALL SATR$$ (K$DTIM,FILNAM,6,ENTRY(21),CODE)
```

Compatibility

SATR$$ has no corresponding old file system routine. It provides a facility (setting the protection bits of a file) formerly available via PRWFIL. SATR$$ can be used on old and new partitions.

1.3.4.12 SAVE$$ -- Save a P300 Memory Image as a File

Function

SAVE$$ is used to save a Prime 300 memory image as a file in the current UFD.

Calling Sequence

```
CALL SAVE$$(VECTOR,NAME,NAMLEN)
```

Parameters

VECTOR A nine word array the user sets up before calling SAVE$$ VECTOR(1) is set to an integer which is the first location in memory to be saved and VECTOR(2) is set to the last location to be saved. The rest of the array is set at the user's option and has the following meaning:

```
VECTOR(3) saved P register
VECTOR(4) Saved A register
VECTOR(5) Saved B register
VECTOR(6) Saved X register
VECTOR(7) Saved Keys
VECTOR(8) not used
VECTOR(9) not used
```

NAME The name of the file to contain the memory image.

NAMLEN The length in characters of NAME.

Notes on usage

SAVE$$ does not have a CODE argument. On an error, an error message is typed and control returns to command level.

Compatibility

SAVE$$ provides the functionality of SAVE extended for long names. SAVE$$ can be used on new and old partitions.

1.3.4.13 SPAS$$ -- Set UFD passwords

Function

SPAS$$ sets the passwords of the current UFD.

Calling Sequence

CALL SPAS$$(OPASS,NPASS,CODE)

Parameters

OPASS — A three word array that contains the password to set as the owner password.

NPASS — A three word array that contains the password to set as the nonwoner password.

CODE — An integer variable set to the return code.

Notes on usage

SPAS$$ requires owner rights to the current UFD. Passwords should not start with a number nor should they contain blanks, commas, =,!,@,{,},[,],(,or).Passwords should not contain lower-case characters but may contain any other characters including control characters.

If the bounce package is invoked (see section 1.6), file unit 16 is used and the current ufd is opened for writing, then closed. The current ufd should not be open on any unit before making this call.

Compatibility

SPAS$$ has no corresponding old file system subroutine. SPAS$$ works on both new and old partitions.

1.3.4.14 SRCH$$ -- Open or Close a File

Function

SRCH$$ is used to connect a file to a file unit (open a file), disconnect a file from a file unit (close a file), delete a file, or check on the existence of a file.

Calling Sequence

```
CALL SRCH$$ (ACTION+REF+NEWFIL,NAME,NAMLEN,FUNIT,TYPE,CODE)
```

Parameters

ACTION A subkey indicating the action to be performed. Possible values are:

K$READ--Open NAME for reading on FUNIT.

K$WRIT--Open NAME for writing on FUNIT.

K$RDWR--Open NAME for reading and writing on FUNIT.

K$CLOS--Close file by NAME or by FUNIT.

K$DELE--Delete file NAME.

K$EXST--Check on existence of NAME.

REF A subkey modifying the ACTION subkey as follows:

K$IUFD--Search for file NAME in the current UFD (this is the default).

K$ISEG--Perform the action specified by ACTION on the file that is a segment directory entry in the directory open on file unit NAME.

K$CACC--Change the access rights of the file already open on FUNIT to ACTION.

NEWFIL A subkey indicating the type of file to create if NAME does not exist. Possible values are:

K$NSAM--New threaded (SAM) file (this is the default).

K$NDAM--New directed (DAM) file.

K$NSGS--New threaded (SAM) segment directory.

K$NSGD--New directed (DAM) segment directory.

Note that it is not possible to generate a new UFD with SRCH$$. Use CREA$$ instead.

NAME The name of the file to be opened. The key OPNCUR can be used to open the current UFD (ACTION keys K$READ, K$WRIT, or K$RDWR only). If REF is K$ISEG, NAME is a file unit from 1 to 16 (1 to 15 under PRIMOS II or PRINET) on which a segment directory is already open.

NAMLEN The length in characters of name.

FUNIT The number (1-16, 1-15 under PRIMOS II or PRINET) of the file unit to be opened or closed.

TYPE An integer variable that is set to the type of the file opened. TYPE is set only on calls that open a file -- it is unmodified for other calls. Possible values of TYPE are:

0	SAM file
1	DAM File
2	SAM Segment Directory
3	DAM Segment Directory
4	UFD

CODE An integer variable set to the return code.

Notes on Usage

The keys REWIND and TRUNCATE of the old SEARCH are now PRWF$$ functions.

Note that it is no longer necessary to call GETERR to obtain the type of the file opened. (Indeed, ERRVEC is no longer set up with the filetype.)

A UFD may be opened only for reading. An attempt to open a UFD for writing will result in an E$NRIT error (error message 'NO RIGHT').

A UFD cannot be deleted unless it is empty. A segment directory cannot be deleted unless it is of length 0. (It can be made to be 0 length by a SGDR$$ call with the MAKSIZ key -- see description of SGDR$$.)

Examples

1) Open new SAM file named RESULTS for output on file unit 2:

```
CALL SRCH$$ (K$WRIT,'RESULTS ',7,2,TYPE,CODE)
```

2) Create new DAM file in the segment directory open on SGUNIT and open for reading and writing on DMUNIT:

```
CALL SRCH$$ (K$RDWR+K$ISEG+K$NDAM,SGUNIT,1,DMUNIT,TYPE,CODE)
```

3) Close and delete the file created in the above call:

```
CALL SRCH$$ (K$CLOS,0,0,DMUNIT,0,CODE)
CALL SRCH$$ (K$DELE+K$ISEG,SGUNIT,0,0,0,CODE)
```

4) See if filename 'MY.BLACK.HEN ' is in current UFD:

```
CALL SRCH$$ (K$EXST+K$IUFD,'MY.BLACK.HEN ',12,0,TYPE,CODE)
IF (CODE.EQ.E$FNTF) CALL TNOU('NOT FOUND ',9)
```

5) Create a new segment directory and a new SAM file as its first entry:

```
CALL SRCH$$ (K$RDWR+K$NSGS,'SEGDIR ',6,UNIT,TYPE,CODE)
CALL SRCH$$ (K$WRIT+K$NSAM+K$ISEG,UNIT,0,7,TYPE,CODE)
```

Compatibility

SRCH$$ provides all the functionality of SEARCH except the REWIND and TRUNCATE functions, which are now provided by PRWF$$. Also, since UFDs cannot be read with PRWF$$, filned or closed.

TYPE An integer variable that is set to the type of the file opened. TYPE is set only on calls that open a file -- it is unmodified for other calls. Possible values of TYPE are:

0	SAM file
1	DAM File
2	SAM Segment Directory
3	DAM Segment Directory
4	UFD

CODE An integer variable set to the return code.

Notes on Usage

The keys REWIND and TRUNCATE of the old SEARCH are now PRWF$$ functions.

Note that it is no longer necessary to call GETERR to obtain the type of the file opened. (Indeed, ERRVEC is no longer set up with the filetype.)

A UFD may be opened only for reading. An attempt to open a UFD for writing will result in an E$NRIT error (error message 'NO RIGHT').

A UFD cannot be deleted unless it is empty. A segment directory cannot be deleted unless it is of length 0. (It can be made to be 0 length by a SGDR$$ call with the MAKSIZ key -- see description of SGDR$$.)

Examples

1) Open new SAM file named RESULTS for output on file unit 2:

```
CALL SRCH$$ (K$WRIT,'RESULTS',7,2,TYPE,CODE)
```

2) Create new DAM file in the segment directory open on SGUNIT and open for reading and writing on DMUNIT:

```
CALL SRCH$$ (K$RDWR+K$ISEG+K$NDAM,SGUNIT,1,DMUNIT,TYPE,CODE)
```

3) Close and delete the file created in the above call:

```
CALL SRCH$$ (K$CLOS,0,0,DMUNIT,0,CODE)
CALL SRCH$$ (K$DELE+K$ISEG,SGUNIT,0,0,0,CODE)
```

4) See if filename 'MY.BLACK.HEN' is in current UFD:

```
CALL SRCH$$ (K$EXST+K$IUFD,'MY.BLACK.HEN',12,0,TYPE,CODE)
IF (CODE.EQ.E$FNTF) CALL TNOU('NOT FOUND',9)
```

5) Create a new segment directory and a new SAM file as its first entry:

```
CALL SRCH$$ (K$RDWR+K$NSGS,'SEGDIR',6,UNIT,TYPE,CODE)
CALL SRCH$$ (K$WRIT+K$NSAM+K$ISEG,UNIT,0,7,TYPE,CODE)
```

Compatibility

SRCH$$ provides all the functionality of SEARCH except the REWIND and TRUNCATE functions, which are now provided by PRWF$$. Also, since UFDs cannot be read with PRWF$$, files can no longer be opened via a K$ISEG through UFD entries. SRCH$$ can be used on old and new partitions.

1.3.4.15 SGDR$$ -- Position and Read Segment Directory Entries

Function

SGDR$$ positions in a segment directory, reads entries, and allows modification of a directory's size.

Calling Sequence

```
CALL SGDR$$ (KEY,FUNIT,ENTRYA,ENTRYB,CODE)
```

Parameters

KEY An integer specifying the action to be performed. Possible values are:

K$SPOS--Move the file pointer of FUNIT to the position given by the value of ENTRYA. Return 1 in ENTRYB if ENTRYA contains a file, return 0 if ENTRYA exists but does not contain a file, return -1 if ENTRYA does not exist (is beyond EOF). If EOF is reached on K$SPOS, the file pointer is left at EOF. The directory must be open for reading or both reading and writing.

K$GOND--Move the file pointer of FUNIT to the end-of-file position and return in ENTRYB the file entry number of the end of the file.

K$GPOS--Return in ENTRYB the file entry number pointed to by the file pointer of FUNIT.

K$MSIZ--Make the segment directory open on FUNIT ENTRYA entries long. The file pointer is moved to the end of file. The directory must be open for both reading and writing

K$MVNT--The entry pointed to by ENTRYA is moved to the entry pointed to by ENTRYB. The ENTRYA entry is replaced with a null pointer. Errors are generated by K$MVNT if there is no file at ENTRYA, if there is already a file at ENTRYB, if either ENTRYA or ENTRYB are at or beyond EOF. The file pointer is left at an undefined position. The directory must be open for both reading and writing.

FUNIT The file unit on which the segment directory is open.

ENTRYA An unsigned 16-bit entry number in the directory, to be interpreted according to KEY.

ENTRYB An unsigned 16-bit integer set or used according to KEY.

CODE An integer variable set to the return code.

Notes on Usage

When using SGDR$$, the segment directory should not be opened for write-only access.

A K$MSIZ call with ENTRYA=0 will cause the directory to have no entries. If the value of ENTRYA is such as to truncate the directory, all entries including and beyond the one pointed to by ENTRYA must be null.

N.B.: When sequentially reading a directory (K$SPOS, ENTRYA = ENTRYA+1, K$SPOS, ...), ENTRYB=-1 indicates the end of the directory, NOT the return code E$EOF. E$EOF will be returned when ENTRYA indicates a position beyond EOF, i.e., the entry following the first K$POS to return ENTRYB=-1.

Examples

1) Read sequentially through the segment directory open on 6:

```
        CURPOS=-1
100     CURPOS=CURPOS+1
        CALL SGDR$$ (K$SPOS,6,CURPOS,RETVAL,CODE)
        IF (RETVAL) 200,300,400 /* BOTTOM, NO FILE, IS FILE
```

2) Make directory open on 2 as big as directory open on 1:

```
        CALL SGDR$$ (K$GOND,1,0,SIZE,CODE)
        IF (CODE.NE.0) GOTO <error handler>
        CALL SGDR$$ (K$MSIZ,2,SIZE,0,CODE)
```

Compatibility

SGDR$$ provides functionality formerly available via PRWFIL. Note, however, that relative positioning is no longer allowed.

SGDR$$ will work on old and new segment directories (i.e., will work on both one-word and two-word entry segment directories).

1.3.4.16 TEXTO$ -- Check Validity of Filename

Function

TEXTO$ checks to see if a filename has valid format.

Calling Sequence

CALL TEXTO$ (NAME,NAMLEN,TRULEN,TEXTOK)

Parameters

NAME An array containing the filename to be checked.

NAMLEN The length of NAME in characters.

TRULEN An integer set to the true number of characters in NAME. TRULEN is valid only if TEXTOK is .TRUE.

TEXTOK A logical variable set to .TRUE. if NAME is a valid filename, else set to .FALSE.

Notes on Usage

TRULEN is the number of characters in NAME preceeding the first blank. If there are no blanks, TRULEN is equal to NAMLEN. The characters valid in filenames are given in Section 1.2.1.1.

Examples

1) Read name from terminal, check for validity, set TRULEN to actual name length:

```
CALL I$AA12 (0,BUFFER,80,$999)
CALL TEXTO$ (BUFFER,32,TRULEN,OK) /* SET TRULEN
IF (.NOT.OK) GOTO <bad-name>
```

Compatibility

TEXTO$ extends the functionality of TEXTOK.

1.3.5 ERROR CODE SUMMARY

The following table summarizes all new file system error codes. Numeric definitions are given in the next section on SYSCOM.

X=>Possible Error O=>Old Partition Only B=>Bounce Package Only

CODE	ATCH$$	CREA$$	GPAS$$	PRWF$$	RDEN$$	SATR$$	SRCH$$	SGDR$$
E$EOF				X	X			X
E$BOF				X	X			X
E$UNOP				X	X		X(1)	X
E$UIUS		B	B			B	X(2)	
E$FIUS		B	B			B	X(2)	
E$BPAR(3)								
E$NATT	X	X	X			X	X(4)	
E$FDFL		O					O(5)	
E$DKFL		X		X(5)			X(5)	X(5)
E$NRIT		X	X			X	X	X
E$FDEL							X	
E$NTUD	X		X				X(4)	
E$NTSD							X(8)	
E$DIRE(3)								
E$FNTF	X		X			X	X(4)	
E$FNTS							X(8)	X
E$BNAM		X					X(4,5)	
E$EXST		X						X
E$DNTE							X	
E$SHUT(3)								
E$DISK(6)	X	X	X	X	X	X	X	X
E$BDAM				X			X	
E$PTRM	X	X	X	X	X	X	X	X
E$BPAS	X(7)							
E$BCOD(3)								
E$BTRN								X
E$OLDP						O		
E$BKEY	X				X	X	X	X
E$BUNT				X	X		X	X
E$BSUN							X(8)	
E$SUNO							X(8)	
E$NMLG								
E$SDER(3)								
E$BUFD(9)	X	X	X		X	X	X	
E$BFTS					X			
E$FITB								X

Notes

1) Possible for K$CACC key only.

2) Possible for all keys but K$EXST.

3) Internal error -- never seen by user program.

4) Possible only on UFD reference keys.

5) Possible only on write operations.

6) An E$DISK (disk I/O) error will always immediately return the program to PRIMOS command level.

7) An E$BPAS (bad password) error will always immediately return the program to PRIMOS command with no UFD attached.

8) Possible only on segment directory reference keys.

9) An E$BUFD (bad UFD) error will be returned only on a RDEN$$ call and a bounced GPAS$$ call. Other subroutines will place the user at PRIMOS command level.

1.4 NEW FILE SYSTEM KEY AND ERROR DEFINITIONS

1.4.1 KEYS.F -- FILE SYSTEM KEY DEFINITIONS

Keys for the new file system calls are defined in two $INSERT files -- KEYS.F for FORTRAN and KEYS.P for PMA -- in the UFD SYSCOM on Volume 1 of the Master Disk. KEYS.F is reproduced here to aid in correlation of the key names with the old file system keys. KEYS.P is equivalent, using EQUs instead of PARAMETERs.

```
C SYSCOM>KEYS.F      MNEMONIC KEYS FOR FILE SYSTEM (FTN)
      NOLIST
C
C     TABSET 6 11 28 69
C
      INTEGER*2 K$READ,K$WRIT,K$POSN,K$TRNC,K$RPOS,K$PRER,K$PREA,
     X    K$POSR,K$POSA,K$CONV,K$RDWR,K$CLOS,K$DELE,K$EXST,
     X    K$IUFD,K$ISEG,K$CACC,K$NSAM,K$NDAM,K$NSGS,K$NSGD,
     X    K$CURR,K$IMFD,K$ICUR,K$SETC,K$SETH,K$ALLD,K$SPOS,
     X    K$GOND,K$MSIZ,K$MENT,K$ENTR,K$SENT,K$GPOS,K$UPOS,
     X    K$PROT,K$DTIM,K$DMPB,K$NRTN,K$SRTN,K$IRTN
C
      PARAMETER
     X
     X /*********************************************************/
     X /*
     X /*
     X /*      KEY DEFINITIONS
     X /*
     X /*
     X /********************** PRWF$$ **********************
     X /*                ****** RWKEY  ******
     X    K$READ = :1,      /* READ
     X    K$WRIT = :2,      /* WRITE
     X    K$POSN = :3,      /* POSITION ONLY
     X    K$TRNC = :4,      /* TRUNCATE
     X    K$RPOS = :5,      /* READ CURRENT POSITION
     X /*                ****** POSKEY ******
     X    K$PRER = :0,      /* PRE-POSITION RELATIVE
     X    K$PREA = :10,     /* PRE-POSITION ABSOLUTE
     X    K$POSR = :20,     /* POST-POSITION RELATIVE
     X    K$POSA = :30,     /* POST-POSITION ABSOLUTE
     X /*                ****** MODE   ******
     X    K$CONV = :400,    /* CONVENIENT NUMBER OF WORDS
     X /*
     X /********************** SRCH$$ **********************
```

```
X /*              ****** ACTION ******
X /* K$READ = :1,     /* OPEN FOR READ
X /* K$WRIT = :2,     /* OPEN FOR WRITE
X    K$RDWR = :3,     /* OPEN FOR READING AND WRITING
X    K$CLOS = :4,     /* CLOSE FILE UNIT
X    K$DELE = :5,     /* DELETE FILE
X    K$EXST = :6,     /* CHECK FILE'S EXISTENCE
X /*              ****** REF    ******
X    K$IUFD = :0,     /* FILE ENTRY IS IN UFD
X    K$ISEG = :100,   /* FILE ENTRY IS IN SEGMENT DIRECTORY
X    K$CACC = :1000,  /* CHANGE ACCESS
X /*              ****** NEWFIL ******
X    K$NSAM = :0,     /* NEW SAM FILE
X    K$NDAM = :2000,  /* NEW DAM FILE
X    K$NSGS = :4000,  /* NEW SAM SEGMENT DIRECTORY
X    K$NSGD = :6000,  /* NEW DAM SEGMENT DIRECTORY
X    K$CURR = :177777,/* CURRENTLY ATTACHED UFD
X /*
X /********************* ATCH$$ *********************
X /*              ****** KEY    ******
X    K$IMFD = :0,     /* UFD IS IN MFD
X    K$ICUR = :2,     /* UFD IS IN CURRENT UFD
X /*              ****** KEYMOD ******
X    K$SETC = :0,     /* SET CURRENT UFD (DO NOT SET HOME)
X    K$SETH = :1,     /* SET HOME UFD (AS WELL AS CURRENT)
X /*              ****** NAME   ******
X    K$HOME = :0,     /* RETURN TO HOME UFD (KEY=K$IMFD)
X /*              ****** LDISK  ******
X    K$ALLD = :100000,/* SEARCH ALL DISKS
X /* K$CURR = :177777,/* SEARCH MFD OF CURRENT DISK
X /*
X /********************* SGDR$$ *********************
X /*              ****** KEY    ******
X    K$SPOS = :1,     /* POSITION TO ENTRY NUMBER IN SEGDIR
X    K$GOND = :2,     /* POSITION TO END OF SEGDIR
X    K$GPOS = :3,     /* RETURN CURRENT ENTRY NUMBER
X    K$MSIZ = :4,     /* MAKE SEGDIR GIVEN NR OF ENTRIES
X    K$MVNT = :5,     /* MOVE FILE ENTRY TO NEW POSITION
X /*
X /********************* RDEN$$ *********************
X /*              ****** KEY    ******
X /* K$READ = :1,     /* READ NEXT ENTRY
X    K$RSUB = :2,     /* READ NEXT SUB-ENTRY
X /* K$GPOS = :3,     /* RETURN CURRENT POSITION IN UFD
X    K$UPOS = :4,     /* POSITION IN UFD
X /*
X /********************* SATR$$ *********************
X /*              ****** KEY    ******
X    K$PROT = :1,     /* SET PROTECTION
X    K$DTIM = :2,     /* SET DATE/TIME MODIFIED
X    K$DMPB = :3,     /* SET DUMPED BIT
X /*
X /********************* ERPR$$ *********************
```

```
X /*                ****** KEY    ******
X     K$NRTN = :0,      /* NEVER RETURN TO USER
X     K$SRTN = :1,      /* RETURN AFTER START COMMAND
X     K$IRTN = :2       /* IMMEDIATE RETURN TO USER
X /*
X /*
X /*************************************************************
 LIST
```

1.4.2 ERRD.F -- ERROR RETURN CODE DEFINITIONS

The definition of ERRD.F is provided here verbatim to ease correlation of the new error names with old file system error codes. ERRD.P (for PMA) provides exactly the same definitions in the form of EQUs. (The two-character codes at the right are the old file system equivalent codes that were found in ERRVEC(1).)

```
C
C SYSCOM>ERRD.F  DEFINE SYSTEM ERROR CODES AS PARAMETERS
C     JPC 15 NOV 76
      NOLIST
C
C DEFINES ALL ERROR CODES
C
      INTEGER*2 E$EOF,E$BOF,E$UNOP,E$UIUS,E$FIUS,E$BPAR,E$NATT,
     X         E$FDFL,E$DKFL,E$NRIT,E$FDEL,E$NTUD,E$NTSD,E$DIRE,
     X         E$FNTF,E$FNTS,E$BNAM,E$EXST,E$DNTE,E$SHUT,E$DISK,
     X         E$BDAM,E$PTRM,E$BPAS,E$BCOD,E$BTRN,E$OLDP,E$BKEY,
     X         E$BUNT,E$BSUN,E$SUNO,E$NMLG,E$SDER,E$BUFD,E$BFTS,
     X         E$FITB,E$NULL
C
      PARAMETER
     X   E$EOF=  1,  /* END OF FILE                          PE */
     X   E$BOF = 2,  /* BEGINNING OF FILE                    PG */
     X   E$UNOP= 3,  /* UNIT NOT OPEN                        PD,SD */
     X   E$UIUS= 4,  /* UNIT IN USE                          SI */
     X   E$FIUS= 5,  /* FILE IN USE                          SI */
     X   E$BPAR= 6,  /* BAD PARAMETER                        SA */
     X   E$NATT= 7,  /* NO UFT ATTACHED                      SL,AL */
     X   E$FDFL= 8,  /* UFD FULL                             SK */
     X   E$DKFL= 9,  /* DISK FULL                            DJ */
     X   E$NRIT=10,  /* NO RIGHT                             SX */
     X   E$FDEL=11,  /* FILE OPEN ON DELETE                  SD */
     X   E$NTUD=12,  /* NOT A UFD                            AR */
     X   E$NTSD=13,  /* NOT A SEGDIR                         -- */
     X   E$DIRE=14,  /* IS A DIRECTORY                       -- */
     X   E$FNTF=15,  /* (FILE) NOT FOUND                     SH,AH */

     X   E$FNTS=16,  /* (FILE) NOT FOUND IN SEGDIR           SQ */
     X   E$BNAM=17,  /* ILLEGAL NAME                         CA */
     X   E$EXST=18,  /* ALREADY EXISTS                       CZ */
     X   E$DNTE=19,  /* DIRECTORY NOT EMPTY                  -- */
     X   E$SHUT=20,  /* BAD SHUTDN (FAM ONLY)                BS */
     X   E$DISK=21,  /* DISK I/O ERROR                       WE */
     X   E$BDAM=22,  /* BAD DAM FILE (FAM ONLY)              SS */
     X   E$PTRM=23,  /* PTR MISMATCH (FAM ONLY)              PC,DC,AC */
     X   E$BPAS=24,  /* BAD PASSWORD (FAM ONLY)              AN */
     X   E$BCOD=25,  /* BAD CODE IN ERRVEC                   -- */
     X   E$BTRN=26,  /* BAD TRUNCATE OF SEGDIR               -- */
     X   E$OLDP=27,  /* OLD PARTITION                        -- */
     X   E$BKEY=28,  /* BAD KEY                              -- */
```

```
      X   E$BUNT=29,  /* BAD UNIT NUMBER                  -- */
      X   E$BSUN=30,  /* BAD SEGDIR UNIT                  SA */
      X   E$SUNO=31,  /* SEGDIR UNIT NOT OPEN             -- */
      X   E$NMLG=32,  /* NAME TOO LONG                    -- */
      X   E$SDER=33,  /* SEGDIR ERROR                     SQ */
      X   E$BUFD=34,  /* BAD UFD                          -- */
      X   E$BFTS=35,  /* BUFFER TOO SMALL                 -- */
      X   E$FITB=36,  /* FILE TOO BIG                     -- */
      X   E$NULL=37   /* (NULL MESSAGE)                   -- */
       LIST
C
C END SYSCOM>ERRD.F
```

1.5 NEW FILE SYSTEM ERROR HANDLING CONVENTIONS

1.5.1 MOTIVATION

All the new file system routines described in the previous section employ new error handling procedures that will slowly be incorporated into other PRIMOS subsystems. The new error handling facilities will not affect existing programs, and only programs using the new file system calls need to be aware of the new error handling.

The new error handling protocol was motivated by the following considerations.

1) Except for a few restricted cases, FORTRAN non-local GOTOs do not work in 64V mode (available since REV. 10).

2) Non-local GOTOs are a violation of good programming practice.

3) Error information in a recursive/reentrant environment must be associated with a particular call, not left in a single static place (e.g., ERRVEC).

1.5.2 THE RETURN CODE PARAMETER

All error codes, formerly placed in ERRVEC, are now returned to the user in a 16-bit user-supplied integer variable. For example, in the call:

```
CALL PRWF$$ (KEY,UNIT,LOC(BFR),NW,POS,RNW,CODE)

C PRWF$$ (KEY,UNIT,LOC(BFR),NW,POS,RNW,CODE)
```

CODE is an integer PRWF$$ sets to the appropriate return code.

CODE can be thought of as a replacement for the (optional) alternate-return argument.

The effect of the old error handling scheme can be achieved through code such as:

```
CALL CREA$$ (NAME,NAMLEN,OPASS,NPASS,CODE)
IF (CODE.NE.0) GOTO 99
```

which would be equivalent to supplying an ALTRTN of $99 in the old scheme (except, of course, that GETERR need not be called to obtain the error code).

N.B.: CODE should always be checked for zero or non-zero to ensure that errors do not go unnoticed.

1.5.3 STANDARD SYSTEM ERROR CODE DEFINITIONS

Standard system error codes are FORTRAN PARAMETER or PMA EQU variables with standardized names. In all cases, zero means no error. Any other value identifies a particular error or exceptional (not necessarily error) condition. All reference to specific code values (other than zero) should be by the standardized names. For convenience, all names are defined in two $INSERT files -- ERRD.F for FORTRAN and ERRD.P for PMA. These files are included in the UFD SYSCOM on Volume 1 of REV. 13 master disk.

1.5.4 NEW ERROR HANDLING ROUTINE

The following routine -- ERRPR$ -- provides all the new error handling facilities.

ERRPR$ -- Print Standard System Error Message

Function

ERRPR$ interprets a return code and, if non-zero, prints a standard message followed by optional user text.

Calling Sequence

```
CALL ERRPR$ (KEY,CODE,TEXT,TXTLEN,NAME,NAMLEN)
```

Parameters

KEY — An integer specifying the action to take subsequent to printing the message. Possible values are:

K$NRTN--Exit to the system, never return to the calling program.

K$SRTN--Exit to the system, return to the calling program following an 'S' command.

K$IRTN—Return immediately to the calling program.

CODE An integer variable containing the return code from the routine that generated the error.

TEXT A message to be printed following the standard error message. TEXT is omitted by specifying both TEXT and TXTLEN as 0.

TXTLEN The length in characters of TEXT.

NAME The name of the program or subsystem detecting or reporting the error. NAME is omitted by specifying both NAME and NAMLEN as 0.

NAMLEN The length in characters of NAME.

Notes on Usage

If CODE is 0, no printing occurs, and ERRPR$ immediately returns to the calling program. The format of the message for non-zero values of CODE is:

<standard text>. <user's TEXT if any> (<NAME if any>)

The system standard text associated with CODE is not preceded by any newlines or blanks and ends with a period. If TXTLEN is greater than zero, this is followed by a blank followed by no more than 64 characters of TEXT. If NAMLEN is greater than zero, this is followed by a blank and no more than 64 characters of NAME enclosed in parentheses. The line is terminated with a newline.

If ERRPR$ is called with the special error code E$NULL, no system message is printed. Other parameters behave normally.

If ERRPR$ is called with an unrecognized value of CODE, the standard system message is 'ERROR=ddddd', where ddddd is the decimal value of CODE. This can be used to display user-defined errors. User defined errors should use codes above 10000.

Examples

1) Following a call to PRWF$$, if CODE=E$UNOP, the call

```
CALL ERRPR$ (K$SRTN,CODE,'DO A STATUS',11,'PRWF$$ ',6)
```

would result in the message:

UNIT NOT OPEN. DO A STATUS (PRWF$$)

2) To print a user-defined error message:

```
CALL ERRPR$ (K$IRTN,10328,'MY MESSAGE',10,0,0)
```

will print:

ERROR=10328. MY MESSAGE

Compatibility

ERRPR$ provides and extends the functionality of PRERR.

1.6 THE BOUNCE PACKAGE

1.6.1 FUNCTIONALITY

The "bounce" package is a set of subroutines that handle new file system calls in circumstances in which the new file system subroutines are not available. The package converts the new file system calls into one or more calls to old file system routines, the effect of which will be equivalent to the new file system calls. Circumstances under which the bounce package is invoked are the following:

1) New file system calls made by a program running under PRIMOS II (DOS), SDOS, or RTOS.

2) A program running under any version of PRIMOS making a new file system call that results in a remote access across the PRINET network.

1.6.2 BOUNCE PACKAGE IMPLEMENTATION RESTRICTIONS

The following restrictions apply to programs using the bounce package:

1) The bounce package, even though it simulates the new file system calls, will not work on new partitions.

2) The bounce package cannot enforce the owner-rights requirement when accessing the current UFD -- only read or write priveledge is required.

3) For calls that may potentially generate more than one error condition, the bounce package is not guarranteed to find the errors in the same order as PRIMOS. For example, a call to SRCH$$ has both a bad filename and a illegal unit number. PRIMOS will return the E$BNAM -- illegal name -- error, while the bounce package will return E$BUNT -- bad unit number.

4) On calls to CREA$$,SPAS$$ GPAS$$, and SATR$$, the bounce package uses file unit 16. Calls to these routines with unit 16 open will cause unpredictable results.

1.6.3 LOADING THE BOUNCE PACKAGE

The bounce package resides in FINLIB (in UFD LIB) and will be correctly loaded by specifying the LOADER 'LIB' command. (The package will not, however, actually be invoked except as noted in 1.6.1 above.)

1.7 SAMPLE PROGRAMS

1.7.1 WRITE SAM FILES

```
C SAMWRT  BIN  29NOV76  PROGRAM TO WRITE A SAM DATA FILE
C
C THE FILE IS 1000 WORDS LONG WRITTEN FROM ARRAY BUFF
C
C RESTRICTIONS: SAMFIL SHOULD NOT EXIST BEFORE RUNNING PROGRAM
C
C
      INTEGER*2 FUNIT1  /* FILE UNIT TO BE USED
      INTEGER*2 SAMFIL  /* FILE TYPE FOR SAM FILE
      INTEGER*2 BUFLNG  /* BUFFER LENGTH
C
      PARAMETER FUNIT1=1, SAMFIL=0, BUFLNG=1000
C
      INTEGER*2 BUFF(BUFLNG)  /* DATA BUFFER
      INTEGER*2 TYPE    /* CONTAINS FILE TYPE RETURNED BY SRCH$$
      INTEGER*2 NMREAD  /* NUMBER WORDS READ OR WRITTEN BY PRWF$$
      INTEGER*2 I
      INTEGER*2 CODE  /* HOLDS ERROR RETURN CODE
C
$INSERT SYSCOM>KEYS.F
C
C
C INITIALIZE BUFFER CONTENTS
      DO 10 I= 1, BUFLNG
        BUFF(I) = I
10    CONTINUE
C
C OPEN A NEW SAM DATA FILE CALLED 'SAMFIL' IN CURRENTLY ATTACHED
C UFD FOR WRITTING ON FILE UNIT FUNIT1
C
C SINCE KEYS.F (KEY DEFINITIONS) DEFINES THE KEYS AS PARAMETERS
C THE USE OF MULTIPLE MNEMONIC KEYS WILL NOT GENERATE MORE CODE
C TAN THE USE OF NUMERIC KEYS. THE USE OF MNEMONIC KEYS IS
C RECOMMENDED AT ALL TIMES.
C
      CALL SRCH$$(K$WRIT+K$NSAM+K$IUFD,'SAMFIL',6,FUNIT1,TYPE,
     X    CODE)
      IF (CODE.NE.0) GO TO 9010
      IF (TYPE .NE. SAMFIL) GO TO 9000   /* ERROR
C
C WRITE 1000 WORDS FROM BUFF INTO THE NEW DATA FILE
C
     CALL PRWF$$(K$WRIT,FUNIT1,LOC(BUFF),BUFLNG,INTL(0),NMREAD,
    X    CODE)
      IF (CODE.NE.0) GO TO 9010
C
C K$CLOS FILE. THIS RELEASES UNIT FUNIT1 FOR RE-USE AND INSURES
```

```
C ALL FILE BUFFERS HAVE BEEN WRITTEN TO DISK.
C NOTE PRIMOS WILL NOT AUTOMATICALLY K$CLOS FILES ON 'CALL EXIT'.
C
9000  CALL SRCH$$(K$CLOS, 0, 0, FUNIT1, 0, CODE)
      IF (CODE.NE.0) GO TO 9010
C
C RETURN TO PRIMOS
C
      CALL EXIT
      END
```

1.7.2 WRITE DAME FILE

```
C DAMWRT  BIN  29NOV76  PROGRAM TO WRITE A DAM DATA FILE
C
C NOTE THAT THE ONLY DIFFERENCE FROM PROGRAM SAMFIL IS THE
C 'NEW FILE' KEY SUPPLIED TO SRCH$$ IN CREATING THE FILE
C
C RESTRICTION: DAMFIL SHOULD NOT EXIST BEFORE RUNNING PROGRAM
C
C
      INTEGER*2 FUNIT1  /* FILE UNIT TO BE USED
      INTEGER*2 DAMFIL  /* FILE TYPE OF DAM DATA FILE
      INTEGER*2 BUFLNG  /* DATA BUFFER LENGTH IN WORDS
C
      PARAMETER FUNIT1=1, DAMFIL=1, BUFLNG=1000
C
      INTEGER*2 BUFF(BUFLNG)  /* DATA BUFFER
      INTEGER*2 TYPE    /* FILE TYPE RETURNED BY SRCH$$
      INTEGER*2 NMREAD  /* NUMBER WORDS READ OR WRITTEN BY PRWF$$
      INTEGER*2 CODE    /* ERROR CODE RETURNED FROM FILE SYSTEM
      INTEGER*2 I
C
$INSERT SYSCOM>KEYS.F
$INSERT SYSCOM>ERRD.F
C
C
C INITIALIZE BUUFFER
C
      DO 10 I = 1, BUFLNG
        BUFF(I) = I
10    CONTINUE
C
C INSURE THAT THE FILE 'DAMFIL' DOES NOT ALREADY EXIST
C
      CALL SRCH$$(K$READ+K$IUFD,'DAMFIL',6,FUNIT1,TYPE,CODE)
      IF (CODE .NE. E$FNTF) GO TO 9000  /* FILE ALREADY EXISTS
C
C OPEN A NEW DAM DATA FILE CALLED 'DAMFIL' IN THE CURRENT
C UFD FOR WRITING ON FILE UNIT FUNIT1 (I.E. CREATE NEW DAM FILE)
C
      CALL SRCH$$(K$WRIT+K$NDAM+K$IUFD,'DAMFIL',6,FUNIT1,TYPE,
     X   CODE)
      IF (CODE.NE.0) GO TO 9010
      IF (TYPE .NE. DAMFIL) STOP   /* WILL NEVER STOP
C
C WRITE THE BUFFER INTO THE FILE
C
     CALL PRWF$$(K$WRIT,FUNIT1,LOC(BUFF),BUFLNG,INTL(0),NMREAD,
     X   CODE)
      IF (CODE.NE.0) GO TO 9010
C
C K$CLOS THE FILE AND EXIT
C
```

```
9000  CALL SRCH$$(K$CLOS, 0, 0, FUNIT1, TYPE, CODE)
      IF (CODE.NE.0) GO TO 9010
      CALL EXIT
C
9010  CALL ERRPR$(K$NRTN,CODE,0,0,0,0)
      END
```

1.7.3 READ A SAM OR DAM FILE

```
C REDFIL  BIN  29NOV76   READ SAM/DAM FILE, PRINT LARGEST INTEGER
C
C THIS PROGRAM SHOWS HOW TO USE THE 'CODE' ERROR RETURN
C MECHANISM AND SUBROUTINE ERRPR$ TO PRINT ERROR MESSAGES.
C
C NOTE THAT PROGRAM DOESN'T CHECK IF THE DATA FILE IS SAM OR DAM.
C TO USER'S PROGRAM, SAM OR DAM FILES ARE FUNCTIONALLY EQUIVALENT
C EXCEPT FOR ACCESS TIME TO RAMDOM POINTS IN THE FILE
C
C RETRICTIONS: NONE
C
C
      INTEGER*2 FUNIT   /* FILE UNIT TO BE USED
      INTEGER*2 DAMFIL  /* TYPE OF DAM DATA FILE
      INTEGER*2 BUFLNG  /* LENGTH OF DATA BUFFER IN WORDS
C
      PARAMETER FUNIT=1, DAMFIL=2, BUFLNG=100
C
      INTEGER*2 BUFF(BUFLNG)  /* DATA BUFFER
      INTEGER*2 TYPE    /* FILE TYPE RETURNED BY SRCH$$
      INTEGER*2 NMREAD  /* NUMBER WORDS READ OR WRITTEN BY PRWF$$
      INTEGER*2 CODE    /* ERROR CODE RETURNED BY FILE SYSTEM
      INTEGER*2 LARGST  /* LARGEST UNSIGNED INTEGER IN FILE
      INTEGER*2 FNAME(16)  /* FILE NAME BUFFER
      INTEGER*2 I,N
C
      INTEGER*4 POSITN  /* 32BIT INTEGER POSITION FOR PRWF$$
C
$INSERT SYSCOM>KEYS.F
$INSERT SYSCOM>ERRD.F
C
C
C INITIALIZE AND GET FILE NAME FROM TERMINAL
C
      LARGST = -32767  /* LARGEST UNSIGNED INTEGER
10    WRITE(1,1000)  /* FORTRAN UNIT 1 IS TERMINAL
1000  FORMAT ('TYPE FILE NAME')
C
      READ(1,1010) (FNAME(I), I=1,16)
1010  FORMAT (16A2)
C
C OPEN FNAME IN CURRENTLY ATTACHED UFD FOR READING ON FILE UNIT 1
C (NOT THE SAME AS FORTRAN UNIT 1). CHECK FOR ERRORS.
C NOTE THAT THE NAME NEED NOT ACTUALLY BE 32 CHARACTERS LONG AS
C TRAILING BLANKS ARE IGNORED.
C
      CALL SRCH$$(K$READ+K$IUFD,FNAME,32,FUNIT,TYPE,CODE)
      IF (CODE .EQ. 0) GO TO 100  /* NO ERRORS
C
C PRINT THE SYSTEM ERROR MSG AND IMMEDIATELY RTRN TO THIS PROGRAM
C IF THE ERROR IS 'FILE NOT FOUND', GET ANOTHER NAME.
```

```
C GIVE UP ON ALL OTHER ERRORS
C
      CALL ERRPR$(K$IRTN, CODE, FNAME, 32, 'REDFIL', 6)
      IF (CODE.EQ.E$FNTF) GO TO 10 /*NOT FOUND-GET ANOTHER NAME
      GO TO 9010   /* ANOTHER TYPE OF ERROR - GIVE UP
C
C THE FILE HAS BEEN OPENED.
C MAKE SURE THE FILE IS NOT A DIRECTORY
C
100   IF (TYPE .GT. DAMFIL) GO TO 9000   /* IS A DIRECTORY
C
C READ AN 'OPTIMAL' NUMBER OF WORDS UP TO BUFLNG WORDS FROM FILE.
C SET LARGST TO THE LARGEST UNSIGNED INTEGER IN THE FILE.
C CHECK FOR END-OF-FILE.
C
30    CALL PRWF$$(K$READ+K$CONV, FUNIT, LOC(BUFF),BUFLNG,
     X                                  INTL(0),NMREAD,CODE)
      IF (CODE .EQ. E$EOF) GO TO 31  /* END-OF-FILE
      IF (CODE .NE. 0) GO TO 9010  /* SOME OTHER ERROR
31    DO 40 I= 1, NMREAD   /* FOR EACH WORD ACTUALLY READ
         IF ((LARGST.LE.0).AND.(BUFF(I).GE.0)) LARGST = BUFF(I)
         IF (LARGST .LT. BUFF(I)) LARGST = BUFF(I)
40    CONTINUE
      IF (CODE .NE. E$EOF) GO TO 30  /* MORE DATA IN FILE
C
C FIND OUT IF THE DATA FILE IS EMPTY
C GET CURRENT FILE POINTER POSITION WHICH IS NOW AT END-OF-FILE.
C IF THE POSITION IS 0, THE FILE IS EMPTY
C
      CALL PRWF$$(K$RPOS, FUNIT, 0, 0, POSITN, NMREAD, CODE)
      IF (CODE .NE. 0) GO TO 9010  /* ERROR
      IF (POSITN .GT. 0) GO TO 50  /* NOT A NULL FILE
      WRITE(1,1030)
1030  FORMAT ('FILE EMPTY')
      GO TO 9000   /* EXIT
C
C FILE NOT EMPTY. PRINT LARGEST INTEGER
C
50    WRITE(1,1020) LARGST
1020  FORMAT ('LARGEST INTEGER IN FILE IS ',I6)
      GO TO 9000  /* EXIT
C
C K$CLOS FILES  EXIT
C PRINT ERROR MESSAGE IF NECESSARY
C
9010  CALL ERRPR$(K$IRTN, CODE, 0, 0, 'REDFIL', 6)
C
9000  CALL SRCH$$(K$CLOS, 0, 0, FUNIT, TYPE, CODE)
      IF (CODE.NE.0) GO TO 9010
      CALL EXIT
      END
```

1.7.4 CREATE A SEGMENT DIRECTORY

```
C CRTSEG  BIN  29NOV76  CREATE A SEGMENT DIRECTORY
C                AND WRITE DATA FILE IN IT
C
C RESTRICTIONS: SEGDIR SHOULD NOT EXIST BEFORE RUNNING PROGRAM
C
C
      INTEGER*2 BUFLNG  /* DATA BUFFER LENGTH
      INTEGER*2 SAMSEG  /* FILE TYPE OF SAM SEGMENT DIRECTORY
      INTEGER*2 SGUNIT  /* FILE UNIT FOR SEGMENT DIRECTORY
      INTEGER*2 FUNIT   /* FILE UNIT FOR DATA FILE
C
      PARAMETER BUFLNG=10, SAMSEG=2, SGUNIT=1, FUNIT=2
C
      INTEGER*2 BUFF(BUFLNG)  /* DATA BUFFER
      INTEGER*2 TYPE    /* FILE TYPE RETURNED BY SRCH$$
      INTEGER*2 NMREAD  /* NUMBER WORDS READ OR WRITTEN BY PRWF$$
      INTEGER*2 I
      INTEGER*2 CODE   /* RETURN CODE STORED HERE
      INTEGER*2 CODEA  /* SCRATCH CODE
C
$INSERT SYSCOM>KEYS.F
$INSERT SYSCOM>ERRD.F
C
C
C INITIALIZE DATA BUFFER CONTENTS
C
      DO 10 I= 1, BUFLNG
        BUFF(I) = I
10    CONTINUE
C
C OPEN A NEW SAM SEGMENT DIRECTORY CALLED 'SAMDIR' IN CURRENTLY
C ATTACHED UFD FOR READING AND WRITING ON FILE UNIT SGUNIT.
C NOTE: SEGDIRS OPEN FOR WRITE ONLY WILL NOT BE HANDLED CORRECTLY
C
      CALL SRCH$$(K$RDWR+K$NSGS+K$IUFD,'SEGDIR',6,SGUNIT,TYPE,
     X    CODE)
      IF (CODE.NE.0) GO TO 9500
      IF (TYPE.NE.SAMSEG) GO TO 9500  /* ERROR--MUST HAVE EXISTED
C
C ENTER A NEW SAM DATA FILE (I.E. OPEN SAM DATA FILE FOR WRITING)
C IN THE JUST CREATED SEGMENT DIRECTORY. THE NEW DATA FILE
C WILL BE ENTRY 0 IN THE SEGMENT DIRECTORY.
C
      CALL SRCH$$(K$WRIT+K$NSAM+K$ISEG,SGUNIT,0,FUNIT,TYPE,CODE)
      IF (CODE.NE.0) GO TO 9500
C
C WRITE THE DATA BUFFER INTO THE JUST CREATED SAM FILE.
C K$CLOS THE DATA FILE.
C
      CALL PRWF$$(K$WRIT,FUNIT,LOC(BUFF),BUFLNG,INTL(0),NMREAD,
     X    CODE)
```

```
      IF (CODE.NE.0) GO TO 9500
      CALL SRCH$$(K$CLOS, 0, 0, FUNIT, 0, CODE)
      IF (CODE.NE.0) GO TO 9500
C
C REPLACE BUFF WITH NEW DATA
C
      DO 20 I= 1, BUFLNG
        BUFF(I) = I * 10
20    CONTINUE
C
C OPEN A DIFFERENT NEW SAM DATA FILE ON FUNIT FOR WRITING
C (I.E. ENTER ANOTHER FILE IN SEGMENT DIRECTORY). THIS IS DONE
C IN TWO STEPS. FIRST THE FILE POINTER OF THE SEGMENT DIR UNIT IS
C POSITIONED TO THE ENTRY NUMBER DESIRED. THE SRCH$$ IS
C CALLED AS ABOVE.
C
      CALL SGDR$$(K$SPOS,SGUNIT, 1, I, CODE)
      IF (CODE.NE.0) GO TO 9500
      IF (I .NE. -1) GO TO 9500  /* ERROR EXIT
C
C NOTE THAT THE SEGMENT DIRECTORY OPEN ON SGUNIT HAS ONLY 1 ENTRY
C (ENTRY 0) AT THIS TIME. THUS, POSITIONING TO ENTRY 1
C WILL POSITION TO END-OF-FILE (NOT BEYOND) AND THE FOLLOWING
C CALL TO SRCH$$ WILL CAUSE THE SEGMENT DIRECTORY TO BE EXTENDED
C IN LENGTH BY ONE ENTRY.
C
      CALL SRCH$$(K$WRIT+K$NSAM+K$ISEG,SGUNIT,0,FUNIT,TYPE,CODE)
      IF (CODE.NE.0) GO TO 9500
C
C WRITE DATA INTO THE SAM FILE  THE K$CLOS THE FILE
C
      CALL PRWF$$(K$WRIT,FUNIT,LOC(BUFF),BUFLNG,INTL(0),NMREAD,
     X    CODE)
      IF (CODE.NE.0) GO TO 9500
      CALL SRCH$$(K$CLOS, 0, 0, FUNIT, 0, CODE)
      IF (CODE.NE.0) GO TO 9500
C
C REPLACE THE BUFFER WITH NEW DATA
C
      DO 30 I= 1, BUFLNG
        BUFF(I) = I * 100
30    CONTINUE
C
C MAKE THE SEGMENT DIRECTORY ITSELF LARGE ENOUGH TO CONTAIN
C 10 ENTRIES. PLACE A SAM FILE IN THE 10TH ENTRY.
C
      CALL SGDR$$(K$MSIZ, SGUNIT, 10, 0, CODE)
      IF (CODE.NE.0) GO TO 9500
C
C THE FILE POINTER ASSOCIATED WITH SGUNIT IS NOW AT END-OF-FILE.
C A CALL TO SRCH$$ WITHOUT FURTHER POSITIONING THE SEGMENT
C DIRECTORY'S FILE POINTER WOULD EXTEND THE SEGMENT DIRECTORY
C AND ENTER THE NEW FILE AS TH 11TH ENTRY. THEREFORE, SGDR$$
```

```
C MUST BE CALLED TO POSITION TO THE 10TH ENTRY.
C
      CALL SGDR$$(K$SPOS, SGUNIT, 10, I, CODE)
      IF (CODE.NE.0) GO TO 9500
      IF (I .NE. 0) STOP  /* FILE CANNOT BE PRESENT
C
      CALL SRCH$$(K$WRIT+K$NSAM+K$ISEG,SGUNIT,0,FUNIT,TYPE,CODE)
      IF (CODE.NE.0) GO TO 9500
      CALL PRWF$$(K$WRIT,FUNIT,LOC(BUFF),BUFLNG,INTL(0),NMREAD,
     X    CODE)
      IF (CODE.NE.0) GO TO 9500
      CALL SRCH$$(K$CLOS, 0, 0, FUNIT, TYPE, CODE)
      IF (CODE.NE.0) GO TO 9500
C
C K$CLOS SEGMENT DIRECTORY  EXIT
C
      CALL SRCH$$(K$CLOS, 0, 0, SGUNIT, TYPE, CODE)
      IF (CODE.NE.0) GO TO 9500
      CALL EXIT
C
C ERROR EXIT. K$CLOS ALL UNITS. PRINT ERROR MESSAGE AND DO NOT
C ALLOW RESTART. E$NULL IS THE NULL SYSTEM ERROR, I.E.,
C NO SYSTEM ERROR MESSAGE IS PRINTED.
C
9500  CALL SRCH$$(K$CLOS, 0, 0, FUNIT, TYPE, CODEA)
      CALL SRCH$$(K$CLOS, 0, 0, SGUNIT, TYPE, CODEA)
      CALL ERRPR$(K$NRTN,CODE,'UNEXPECTED ERROR',16,'CRTSEG',6)
C
      END
```

1.7.5 READ A LOGICAL RECORD FROM A FILE

```
C RDLREC  EIN  29NOV76  READ A LOGICAL RECORD FROM A FILE
C
C PROGRAM READS LOGICAL RECORD 'N' FROM A FILE CONSISTING
C OF FIXED LENGTH RECORDS
C
C IN THIS PROGRAM, THE FILE ACCESSED IS CONSIDERED TO CONTAIN AN
C UNLIMITED NUMBER OF LOGICAL RECORDS. EACH RECORD CONTAINS 'M'
C WORDS. THE PROGRAM READS AND PRINTS TO THE TERMINAL THE
C CONTENTS OF RECORD NUMBER N AS M INTEGERS. THE FIRST RECORD
C OF A FILE IS RECORD NUMBER 0 (ZERO).
C NOTE THAT A LOGICAL RECORD IS MERELY A GROUPING OF WORDS IN A
C FILE. THE LOGICAL RECORD SIZE HAS NO RELATION TO THE PHYSICAL
C RECORD SIZE OF THE DISK.
C
C RESTRICTIONS:
C   1. RECORD SIZE MUST BE BETWEEN 1 AND BUFFER LENGTH
C   2. RECORD NUMBER MUST BE BETWEEN 0 AND 32767
C   3. THE RECORD MUST BE IN THE FILE
C   4. THE FILE MUST PREVIOUSLY EXIST
C   5. THE FILE MUST BE A DATA FILE (SAMFIL OR DAMFIL)
C
C
      INTEGER*2 FUNIT1  /* PRIMOS FILE UNIT USED FOR DATA FILE
      INTEGER*2 BUFLNG  /* LENGTH OF DATA BUFFER
C
      PARAMETER FUNIT1=1, BUFLNG=1000
C
      INTEGER*2 BUFF(BUFLNG)  /* DATA BUFFER
      INTEGER*2 FNAME(16)  /* FILE NAME BUFFER
      INTEGER*2 RECSIZ  /* NUMBER WORDS IN A LOGICAL RECORD
      INTEGER*2 RECNUM  /* LOGICAL RECORD NUMBER
      INTEGER*2  TYPE    /* FILE TYPE RETURNED BY SRCH$$
      INTEGER*2 NMREAD  /* NUMBER WORDS READ, RETURNED BY PRWF$$
      INTEGER*2 CODE    /* ERROR STATUS RETURNED BY FILE SYSTEM
      INTEGER*2 I
C
      INTEGER*4 POSITN  /* 32BIT WORD NR USED AS POS TO PRWF$$
C
C
$INSERT SYSCOM>KEYS.F
$INSERT SYSCOM>ERRD.F
C
C
C ASK FOR FILE NAME
C
10    WRITE(1,1000)   /* FORTRAN UNIT 1 IS TTY
1000  FORMAT ('TYPE FILE NAME')
C
C READ FILE NAME
C
      READ(1,1010) (FNAME(I),I=1,16)
```

```
1010  FORMAT (16A2)
C
C OPEN FNAME IN CURRENT UFD FOR READING ON FILE UNIT FUNIT1
C
      CALL SRCH$$(K$READ+K$IUFD, FNAME, 32, FUNIT1, TYPE, CODE)
      IF (CODE.NE.0) GO TO 1000
C
C ASK FOR LOGICAL RECORD SIZE
C
20    WRITE(1,1020)
1020  FORMAT ('TYPE RECORD SIZE')
      READ(1,1030) RECSIZ
1030  FORMAT (I6)
      IF (RECSIZ .GE. 1 .AND. RECSIZ .LE. BUFLNG) GO TO 30
      WRITE(1,1040)
1040  FORMAT ('BAD RECORD SIZE')
      GO TO 20
C
C ASK FOR RECORD NUMBER. FIRST RECORD IS NUMBERED 0 (ZERO)
C
30    WRITE(1,1050)
1050  FORMAT ('TYPE RECORD NUMBER')
      READ (1,1030) RECNUM
      IF (RECNUM .GE. 0) GO TO 35
      WRITE(1,1051)
1051  FORMAT ('BAD RECORD NUMBER')
      GO TO 30
C
C CALCULATE THE 32-BIT WORD NUMBER OF THE FIRST WORD IN THE
C DESIRED RECORD. NOTE THAT IF BOTH RECSIZ AND RECNUM ARE BOTH
C POSITIVE 16BIT NUMBERS, THE 32BIT WORD NUMBER MUST ALSO BE
C POSITIVE.
C
C POSITIONING MAY BE DONE TO AN ABSOLUTE WORD NUMBER OR RELATIVE
C TO THE CURRENT POSITION. SINCE A JUST OPENED FILE IS ALWAYS
C POSITIONED TO TOP-OF-FILE AND THE CALCULATED WORD NUMBER WILL
C NEVER BE NEGATIVE, THE ARGUMENT FOR POSITION TO PRWF$$ WILL
C BE THE SAME FOR BOTH CALLS IN THIS PROGRAM.
C
35    POSITN=INTL(RECSIZ)*INTL(RECNUM)   /* POSITN IS INTEGER*4
      IF (POSITN .GT. 32767) GO TO 100  /* ABSOLUTE POSITIONING
C
C RECORD LESS THAN 32767 WORDS FROM THE BEGINNING, USE RELATIVE
C POSITIONING.
C NOTE THAT ABSOLUTE POSITIONING COULD HAVE BEEN USED FOR A
C RECORD ANYWHERE IN THE FILE, NOT JUST FOR THOSE RECORDS
C BEYOND WORD 32767. RELATIVE IS SHOWN HERE ONLY FOR EXAMPLE.
C
C NOTE ALSO THAT RELATIVE POSITIONING COULD BE USED TO POSITION
C TO ANY WORD IN THE FILE, GIVEN THE RESTICTIONS ON RECSIZ AND
C RECNUM.
C
C WHEN REL POSITIONING IS USED, THE POS ARGUMENT (POSITN HERE)
```

```
C IS CONSIDERED TO BE A SIGNED 32-BIT INTEGER.
C
      CALL PRWF$$(K$READ+K$PRER,FUNIT1,LOC(BUFF),RECSIZ,POSITN,
     X                                            NMREAD, CODE)
      GO TO 200  /* SKIP OVER ABSOLUTE POSITION EXAMPLE
C
C RECORD IS MORE THAN 32767 WORDS FROM THE BEGINNING OF FILE, USE
C ABSOLUTE POSITIONING.
C
C WHEN ABSOLUTE POSITIONING IS USED, POSITION ARGUMENT (POSITN)
C IS CONSIDERED TO BE AN SIGNED 32-BIT INTEGER.
C NOTE THAT THE E$BOF ERROR (BEGINNING OF FILE) CAN OCCUR.
C
100   CALL PRWF$$(K$READ+K$PREA,FUNIT1,LOC(BUFF),RECSIZ,POSITN,
     X                                            NMREAD, CODE)
C
200   IF (CODE .NE. 0) GO TO 300  /* ERROR DETECTED
C
C HAVE READ RECORD, NOW TYPE IT.
C
      WRITE(1,1060) RECNUM,RECSIZ
1060  FORMAT('RECORD ',I6,' CONTAINS ',I6,' ENTRIES AS FOLLOWS')
      WRITE(1,1070)  (BUFF(I), I=1,RECSIZ)
1070  FORMAT (10I7)
C
C RETURN TO DOS AFTER CLOSING THE FILE
C
250   CALL SRCH$$(K$CLOS, 0, 0, FUNIT1, TYPE, CODE)
      IF (CODE.NE.0) GO TO 1000
      CALL EXIT
      GO TO 10  /* START COMMAND RESTARTS PROGRAM
C
C ERROR WHILE ATTEMPTING TO READ THE RECORD
C
300   CALL ERRPR$(K$IRTN, CODE, 0, 0, 'RDLREC', 6)
      IF (CODE .NE. E$EOF) GO TO 250  /* EXIT IF NOT END-OF-FILE
C
C END-OF-FILE REACHED.
C REWIND FILE AND TRY AGAIN
C
      CALL PRWF$$(K$POSN+K$PREA,FUNIT1,0,0,INTL(0),NMREAD,
     X    CODE)
      IF (CODE.NE.0) GO TO 1000
      GO TO 20
C
1000  CALL ERRPR$(K$NRTN,CODE,0,0,0,0)
      END
```

1.7.6 Read File in Segment Directory

```
C READSEG  BIN  29NOV76  READ FILE IN A SEGMENT DIRECTORY
C
C THIS PROGRAM READS FILE NUMBER N IN  SEGMENT DIRECTORY AND
C TYPES WORD NUMBER M IN THAT FILE. THE FIRST FILE IN THE
C DIRECTORY IS FILE NUMBER 0. THE FIRST WORD IN THE FILE IS
C WORD NUMBER 0.
C
C RESTRICTIONS:
C   1. THE SEGMENT DIRECTORY FILE MUST EXIST
C   2. THE FILE NUMBER MUST BE BETWEEN 0 AND 32767
C   3. THE FILE MUST BE IN THE SEGMENT DIRECTORY
C   4. THE WORD NUMBER MUST BE BETWEEN 0 AND 32767
C   5. THE WORD MUST BE IN THE FILE.
C
C
      INTEGER*2 FUNIT   /* PRIMOS FILE UNIT FOR DATA FILE
      INTEGER*2 SGUNIT  /* PRIMOS FILE UNIT FOR SEGMENT DIRECTORY
      INTEGER*2 SAMSEG  /* FILE TYPE OF SAM SEGMENT DIRECTORY
      INTEGER*2 DAMSEG  /* FILE TYPE OF DAM SEGMENT DIRECTORY
C
      PARAMETER FUNIT=2, SGUNIT=1, SAMSEG=2, DAMSEG=3
C
      INTEGER*2 BUFF    /* DATA BUFFER
      INTEGER*2 SEGDIR(16)  /* NAME OF SEGMENT DIRECTORY BUFFER
      INTEGER*2 FILNUM  /* FILE NR (ENTRY NR) OF FILE IN SEGDIR
      INTEGER*2 WRDNUM  /* WORD NUMBER IN DATA FILE TO BE READ
      INTEGER*2 CODE    /* ERROR CODE RETURNED BY FILE SYSTEM
      INTEGER*2 TYPE    /* FILE TYPE RETURNED BY SRCH$$
      INTEGER*2 NMREAD  /* NR WORDS READ/WRITTEN/RTRNED BY PRWF$$
      INTEGER*2 I
C
$INSERT SYSCOM>KEYS.F
$INSERT SYSCOM>ERRD.F
C
C
C INSURE FILE UNITS TO BE USED ARE K$CLOSD
C ASK FOR AND READ SEGMENT DIRECTORY NAME FROM TERMINAL
C
10    CALL SRCH$$(K$CLOS, 0, 0, SGUNIT, 0, CODE)
      IF (CODE.NE.0) GO TO 100
      CALL SRCH$$(K$CLOS, 0, 0, FUNIT, 0, CODE)
      IF (CODE.NE.0) GO TO 100
      WRITE(1,1000)
1000  FORMAT ('TYPE SEGMENT DIRECTORY NAME')
      READ (1,1010) (SEGDIR(I), I=1,16)
1010  FORMAT (16A2)
C
C OPEN THE SEGMENT DIRECTORY FOR READING ON SGUNIT
C
      CALL SRCH$$(K$READ+K$IUFD, 'SEGDIR', 6, SGUNIT, TYPE, CODE)
      IF (CODE.NE.0) GO TO 100
```

```
C
C TYPE CONTAINS THE FILE TYPE OF THE FILE JUST OPENED.
C MAKE SURE THE FILE IS EITHER A SAM OR DAM SEGMENT DIRECTORY.
C ALLOWABLE TYPE VALUES ARE 2 AND 3.
C
      IF (TYPE .EQ. SAMSEG) GO TO 20
      IF (TYPE .EQ. DAMSEG) GO TO 20
C
C NOT A SEGMENT DIRECTORY - TRY AGAIN
C
      WRITE(1,1020)
1020  FORMAT('FILE IS NOT A SEGMENT DIRECTORY')
      GO TO 10
C
C ASK FOR FILE (ENTRY) NUMBER IN SEGMENT DIRECTORY
C
20    WRITE(1,1030)
1030  FORMAT ('TYPE FILE NUMBER')
      READ (1,1040) FILNUM
1040  FORMAT (I6)
      IF (FILNUM .LT. 0) GO TO 20
C
C ASK FOR WORD NUMBER IN DATA FILE TO READ
C
30    WRITE(1,1035)
1035  FORMAT ('TYPE WORD NUMBER')
      READ (1,1040) WRDNUM
      IF (WRDNUM .LT. 0) GO TO 30
C
C TRY TO POSITIONIO WORD NUMBER IN THE SEGMENT DIRECTORY.
C IF END-OF-FILE REACHED, FILE IS NOT IN SEGMENT DIRECTORY.
C SGDR$$ RETURNS THE VALUE 1 IN THE 4TH ARGUMENT (TYPE) IF A
C FILE IS ENTERED IN THE ENTRY POSITION. THIS PROGRAM DOES NOT
C CHECK THE VALUE, SINCE SRCH$$ WILL RETURN THE PROPER ERROR CODE
C (E$FNTS - FILE NOT FOUND IN SEGMENT DIRECTORY) ANYHOW.
C
      CALL SGDR$$(K$SPOS, SGUNIT, FILNUM, TYPE, CODE)
      IF (CODE .EQ. E$EOF) CODE = E$FNTS  /* FILE NOT FOUND
      IF (CODE .NE. 0) GO TO 100
C
C OPEN FILE IN SEGMENT DIRECTORY FOR READING
C
      CALL SRCH$$(K$READ+K$ISEG,SGUNIT,0,FUNIT,TYPE,CODE)
      IF (CODE .NE. 0) GO TO 100
C
C PRINT THE WORD, K$CLOS THE FILES, AND RETURN TO PRIMOS
C
      WRITE(1,1050) WRDNUM,FILNUM,(SEGDIR(I), I= 1,16),BUFF
1050  FORMAT ('WORD',I6,' OF FILE (',I6,') IN ',16A2,
     X  'CONTAINS',I6)
50    CALL SRCH$$(K$CLOS, 0, 0, FUNIT, 0, CODE)
      CALL SRCH$$(K$CLOS, 0, 0, SGUNIT, 0, CODE)
      CALL EXIT
```

```
      GO TO 10  /* START COMMAND RE-STARTS PROGRAM
C
C
C COMMON ERROR HANDLER
C NOTE THAT THE NEW FILE SYS PROPERLY DIFFERENTIATES THE VARIOUS
C ERRORS WHICH FORMERLY WERE GROUPED UNDER OLD ERROR CODE 'SQ'
C
100   IF (CODE.EQ.E$FNTS) GO TO 110 /* FILE NOT FOUND IN SEGDIR
      IF (CODE .EQ. E$EOF ) GO TO 120   /* END-OF-FILE
      CALL ERRPR$(K$IRTN,CODE,0,0,'REDSEG',6) /* PRINT ERROR MSG
      GO TO 50   /* K$CLOS FILES  EXIT
C
C FILE NOT FOUND IN SEGMENT DIRECTORY
C LET THE USER TRY AGAIN
C
110   WRITE(1,1060) FILNUM, (SEGDIR(I), I=1, 16)
1060  FORMAT ('FILE (',I6,') NOT FOUND IN ',16A2)
      GO TO 10  /* RE-TRY
C
C END-OF-FILE
C CODE WILL CONTAIN E$EOF ONLY WHILE TRYING TO READ
C THE DATA FILE. ALLOW RE-TRY.
C
120   WRITE(1,1070) WRDNUM,FILNUM,(SEGDIR(I),I=1,16)
1070  FORMAT ('WORD',I6,' NOT IN FILE (',I6,') IN ',16A2)
      GO TO 10  /* RE-TRY
C
      END
```

1.8 INTERNAL FILE SYSTEM FORMATS

The following describes the internal formats of all disk records for both the old and new file system. These have been collected together for ease in noting the changes that have been made. User programs will normally have no need to refer to the internal file system formats. All numbers are decimal unless preceded by a ':'. Where possible, field names are the same as those used by the internal file system routines.

1.8.1 DSKRAT FORMATS

1.8.1.1 DSKRAT Format -- Old Partitions

0	5	NUMBER OF WORDS IN DSKRAT HEADER = 5
1	RECSIZ	DISK RECORD SIZE (448 or 1040)
2	NMRECS	NUMBER RECORDS IN PARTITION
3	UNUSED	UNUSED
4	NHEADS	NUMBER HEADS IN PARTITION
5	DATA	START OF DKSRAT DATA (ONE BIT/RECORD)
	....	

1.8.1.2 DSKRAT Format -- New Partitions

0	8	NUMBER WORDS IN HEADER = 8
1	RECSIZ	RECORD SIZE
2	NMRECS	NUMBER RECORDS IN PARTITON (TWO WCRDS)
4	NHEADS	NUMBER HEADS IN PARTITION
5	RESERVED	RESERVED
6	RESERVED	RESERVED
7	RESERVED	RESERVED
8	DATA	START OF DSKRAT DATA (ONE BIT/RECORD)
	....	

1.8.2 RECORD HEADER FORMATS

Note: record header formats are the same for new and old partitions. The format of a record header is a function of the physical record size.

1.8.2.1 Record Header Format -- 448-Word Records

0	REKCRA	RECORD ADDRESS (OF THIS RECORD)
1	REKBRA	RA OF DIRECTORY ENTRY OR FIRST RECORD
2	REKFPT	RA OF NEXT SEQUENTIAL RECORD
3	REKBPT	RA OF PREVIOUS RECORD
4	REKCNT	NUMBER DATA WORDS IN FILE
5	REKTYP	TYPE OF THIS FILE
6	REKLVL	INDEX LEVEL FOR NEW PARTITION DAM FILES
7	RESERVED	RESERVED

1.8.2.2 Record Header Format -- 1040-Word Records

0	REKCRA	RECORD ADDRESS OF THIS RECORD (TWO WORDS)
2	REKBRA	BEGINNING RECORD ADDRESS (TWO WORDS)
4	REKCNT	NUMBER DATA WORDS IN THIS RECORD
5	REKTYP	TYPE OF THIS FILE
6	REKFPT	RA OF NEXT SEQUENTIAL RECORD (TWO WORDS)
8	REKBPT	RA OF PREVIOUS RECORD (TWO WORDS)
10	REKLVL	INDEX LEVEL FOR NEW PARTITION DAM FILES
11		
	RESERVED	RESERVED (FIVE WORDS)
15		

Notes

1) All disks except Storage Module have 448-word records. Storage Modules have 1040-word records.

2) The BRA of the first record in a file points to the beginning record address of the directory in which the file entry appears. In all other records, the BRA points to the first record of the file.

3) FORPTR contains the address of the next sequential record in the file or 0 if it is the last record in the file.

4) BACPTR contains the address of the previous record in sequence or 0 if it is the first record in the file.

5) FILTYP is valid only in the first record of a file. Legal values are:

0	SAM File
1	DAM File
2	SAM Segment Directory
3	DAM Segment Directory
4	User File Directory (UFD)

6) If the file is the record zero bootstrap (BOOT) or the disk record availability table (DSKRAT or volume name) and the disk has a 1040 record size (Storage Module), bit 1 (:100000) of FILTYP will be set.

7) DAM files on new partitions are organized somewhat differently from the above -- see Section 1.8.5.

1.8.3 UFD HEADER AND ENTRY FORMATS

1.8.3.1 Old UFD Header Format

0	8	SIZE OF HEADER -- 8 WORDS
1	OPASSW	OWNER PASSWORD (THREE WORDS)
4	NPASSW	NON-OWNER PASSWORD (THREE WORDS)
7	RESERVED	RESERVED

1.8.3.2 New UFD Header Format

0	ECW	ECW (SEE NOTE 1 BELOW)
1	OPASSW	OWNER PASSWORD (THREE WORDS)
4	NPASSW	NON-OWNER PASSWORD (THREE WORDS)
7	RESERVED	RESERVED (SIXTEEN WORDS)
23		

1.8.3.3 Old UFD Entry Format

```
 0 |  BRA   |  BEGINNING RECORD ADDRESS
 1 |  FILE  |  FILENAME (THREE WORDS)
   |        |
   |  NAME  |
 4 | SPACES |  TWO BLANKS FOR NAME EXPANSION (RESERVED)
 5 | PROTEC |  PROTECTION (OWNER/NON-OWNER)
```

Notes

In an old UFD, the high-order eight bits of PROTEC are the owner rights stored in complemented form (0=>owner has right). The low-order eight bits are non-owner protection, stored in true form (0=>no right). On creation, PROTEC=0. After a 'PROT 7 0', PROTEC=:174000.

1.8.3.4 New UFD Entry Format

```
 0 |  ECW   |  ENTRY CONTROL WORD (TYPE/LENGTH)
 1 |  ERA   |  BEGINNING RECORD ADDRESS (TWO WORDS)
   |        |
 3 |RESERVED|  RESERVED (THREE WORDS)
   |        |
   |        |
 6 | PROTEC |  PROTECTION (OWNER/NON-OWNER)
 7 |RESERVED|  RESERVED FOR FUTURE USE
 8 | DATMOD |  DATE LAST MODIFIED (YYYYYYYYMMMMDDDDD)
 9 | TIMMOD |  TIME LAST MODIFIED (SECONDS-SINCE-MIDNIGHT/4)
10 | FILTYP |  FILETYPE
11 |  SCW   |  SUBENTRY CONTROL WORD FOR FILENAME
12 |F       |
   |  I     |
   |    L   |
   |      E |
   |  ...   |  FILENAME (1-16 WORDS, BLANK PADDED)
   |N       |
   |  A     |
   |    M   |
 N |      E |
```

Notes

1) The Entry Control Word (ECW) consists of two eight-bit subfields. The top eight bits indicate the type of the following entry as follows:

0	Old UFD Header
1	New UFD Header
2	Vacant Entry
3	New UFD File Entry

The low-order eight bits give the size of the entry including the ECW itself.

2) The bits in PROTEC are stored in true form (0=> no right) for both owner and non-owner fields.

3) The file type field is as before (see Old Record Header Format) with following additional bits:

BIT	MEANING WHEN BIT IS ON
1	File has 16-word header (DSKRAT and BOOT only).
4	Special file (BOOT, DSKRAT, MFD, BADSPT).

4) The Subentry Control Word (SCW) consists of two eight-bit subfields. The top 8 bits are 0, indicating subentry type 0. The low-order 8 bits give the size of the subentry including the SCW itself.

5) N.B.: UFD entries are reused by the new file system. This means that a new entry will not necessarily appear at the end of the UFD.

1.8.4 SEGMENT DIRECTORY FORMATS

1.8.4.1 Old Segment Directory Format

0	BRA0	BRA OF FIRST ENTRY IN DIRECTORY
1	BRA1	BRA OF SECOND FILE
2	0000	NULL ENTRY
	....	
N	BRAn	BRA OF LAST FILE IN DIRECTORY

1.8.4.2 New Segment Directory Format

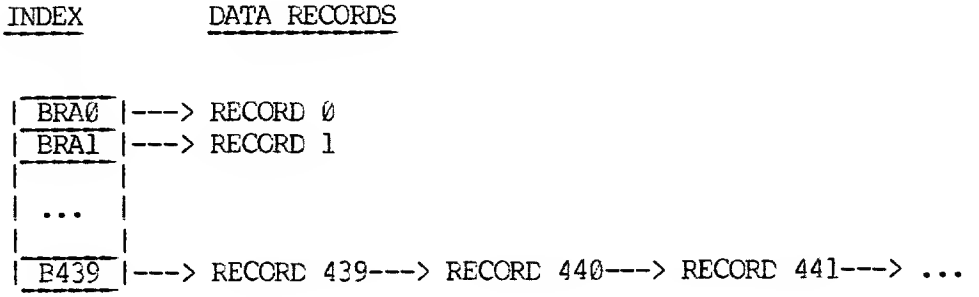
0	BRA0	BRA OF FIRST FILE IN DIRECTORY (TWO WORDS)
2	BRA1	BRA OF SECOND FILE IN DIRECTORY (TWO WORDS)
	0000	NULL ENTRY (TWO WORDS)
	0000	
	....	
2N	BRAn	BRA OF LAST FILE IN DIRECTORY (TWO WORDS)

Notes

The only difference between old and new directories is that each entry has been expanded to two words. A null entry in a new directory is a 32-bit 0.

1.8.5 DAM FILE ORGANIZATION

In old-style DAM files, the first physical record of the file was reserved to be an index to the first 440 or 1024 (depending on physical record size) records in the file. When this index was filled, however, access to subsequently added records became sequential. For example, in the file shown below, records 0-439 can be accessed directly. Records 440 and above must be searched for sequentially starting with record 439.



The major difference between new and old DAM files is that the index is not limited to a single record, but can grow into a multi-level tree. (Also, since pointers are now two words each, each index record holds half the number of pointers in old index records.) An index can grow to any size, and any data record can be directly accessed. The following paragraphs explain how this multi-level index is built.

The handling of a DAM file on a new partition is identical to that on an old partition up to the point at which the index record is full and another record is to be added to the file. At this point the following actions take place.

1) Three new records are obtained from the file system. One of these records is to be the new data record, the other two are used to construct the second index level.

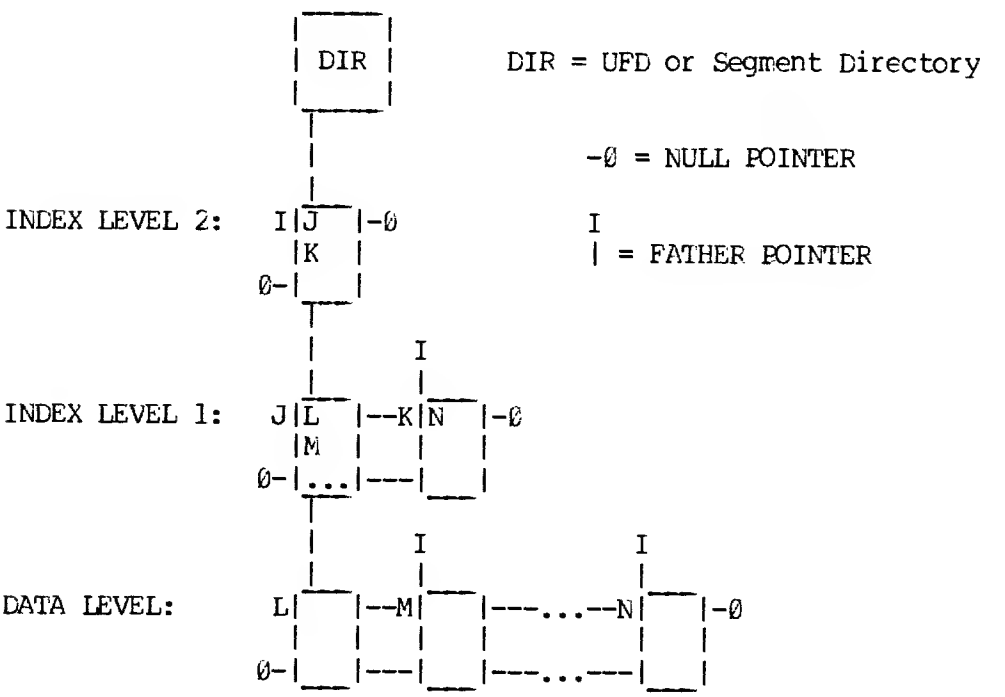
2) The index entries from the full index record are copied into one of the other new records. This record is to become the first index record of the new index level.

3) The old index record is reinitialized to contain two pointers to the two index records on the new level.

4) The other new index record is initialized with a single entry pointing to the new data record.

5) Forward, backward, and father pointers are set up as shown in the diagram below.

At this point, the creation of the new index level is complete. Note that the BRA in the directory entry for the DAM file still points to the original index record, which is now the top of a two-level index.



The DIR entry points to the original index record (record 'I'), which now contains just pointers to records 'J' and 'K' -- the two records on the index level just created. Record 'J' contains the data record pointers originally in 'I' -- 'L', 'M', etc. Record 'K' contains a single pointer to the newly created data record 'N'.

Once an index level is created, it is never deleted until the file itself is deleted -- there will alg paragraphs explain how this multi-level index is built.

The handling of a DAM file on a new partition is identical to that on an old partition up to the point at which the index record is full and another record is to be added to the file. At this point the following actions take place.

1) Three new records are obtained from the file system. One of these records is to be the new data record, the other two are used to construct the second index level.

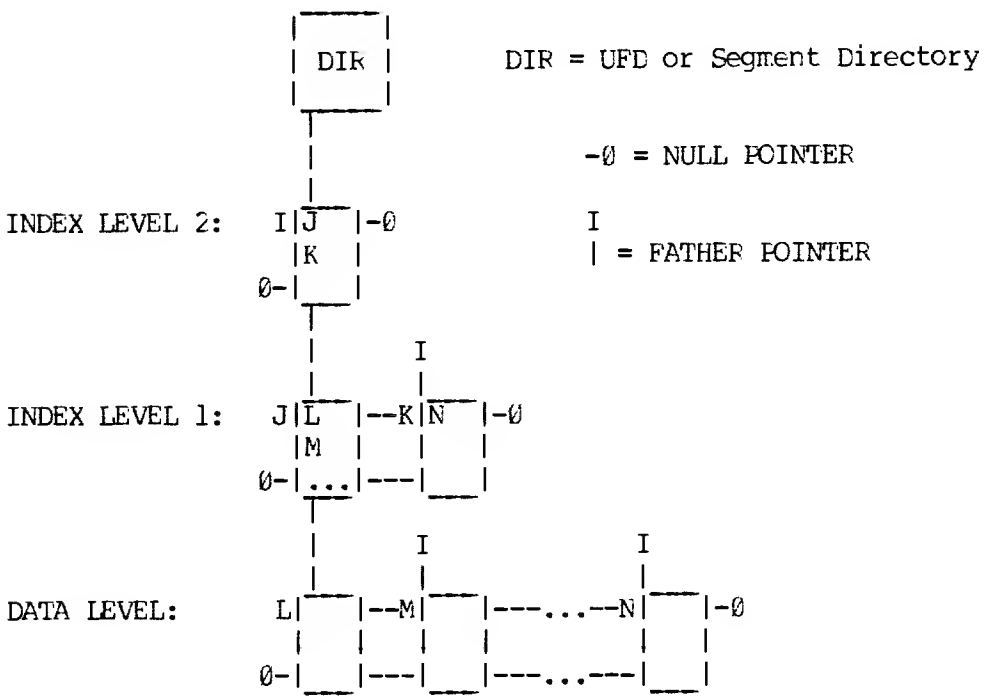
2) The index entries from the full index record are copied into one of the other new records. This record is to become the first index record of the new index level.

3) The old index record is reinitialized to contain two pointers to the two index records on the new level.

4) The other new index record is initialized with a single entry pointing to the new data record.

5) Forward, backward, and father pointers are set up as shown in the diagram below.

At this point, the creation of the new index level is complete. Note that the BRA in the directory entry for the DAM file still points to the original index record, which is now the top of a two-level index.



The DIR entry points to the original index record (record 'I'), which now contains just pointers to records 'J' and 'K' -- the two records on the index level just created. Record 'J' contains the data record pointers originally in 'I' -- 'L', 'M', etc. Record 'K' contains a single pointer to the newly created data record 'N'.

Once an index level is created, it is never deleted until the file itself is deleted -- there will always be at least one record on each level. If the file is empty, there will be exactly one record on each index level. This is to avoid undue thrashing when records are being added and deleted near the threshold of an index's capacity. (Note that the overhead of the "unnecessary" levels is only one record per level.)

PART 2

FUTIL, REV. 12 & 13

2.1 INTRODUCTION

FUTIL has been completely revamped for rev 12 to use new file system calls exclusively, thus allowing it to work on both old and new partitions. Because of the new file system, the operation of FUTIL has changed in some minor ways. In addition, several new features have been added in accordance with the philosophy that FUTIL is a general file system utility and not just a copy and delete program.

The minor functional differences are:

1. To enable listing and copying from write protected disks and to avoid updating date/time modified (DTM) stamps, FUTIL will not attempt to change access rights to files on these operations. Therefore, if any files or directories have even owner access rights (i.e. no read rights), FUTIL will report a "NO RIGHT" error. The user does, however, have the ability to force FUTIL to read files on LISTF and COPY operations, but so doing will update all DTM's on listed or copied directories and further will fail on write protected disks.

2. Since the new file system allows up to 32 character names, all file names must be typed exactly as they are. For example, "DELETE B_ABCDE" would have deleted the file "B_ABCD" at rev 11, but will report "NOT FOUND" at rev 12. Of course, on a new partition, the file "B_ABCDE" will be deleted and, if present, "B_ABCD" will not be deleted.

3. At rev 11, some problems existed in the interaction of ATTACH, TO, and FROM. These problems have been either fixed or clarified for rev 12. In short, both TO and FROM will not affect the other and neither will affect the ATTACH point (i.e. HOME UFD). ATTACH, however, will reset any FROM or TO name beginning with "*" to be simply "*", thus avoiding transferring improper tree names to a new home UFD. Absolute TO and FROM names (i.e. those beginning with a name rather than *) will not be affected by ATTACH.

4. At rev 11, FUTIL would always abort processing upon encountering an error. Because of this, it was possible to end up with partially copied trees, for example, which could o only be completed by redoing the entire operation. At rev 12, FUTIL will report any error conditions, and, with the exception of DISK FULL on copies, continue the operation. As an example of the utility of continuing on errors, it is now possible to UFDDEL all unprotected files and directories while leaving the protected ones intact.

5. Segment directories in the new file system are addressed in terms of entry number rather than record number, word number. Therefore, the syntax of names within segment directories has been changed from (rec, word) to (entry). Thus, the fifth entry in a segment directory is (5) and last entry is (65534). Note that (65535) is not a legal entry as the maximum size of a directory is 65535 including the entry (0).

The new features added, in summary, are:

1. Specification of LISTF output file name (LISTSAVE)
2. Scanning for files of a given name (SCAN)
3. Conditional file delete on prefix match (CLEAN)
4. Mode for forcing access rights on LISTF and copies on "from" directories (FORCE)
5. Ability to create new, empty UFD's on "to" directory (CREATE)
6. Ability to protect a file or directory (PROTECT), files and directories, to any depth (UFDPRO), and an entire sub-tree s

any depth (UFDPRO), and an entire sub-tree structure (TREPRO)

7. Up to 32 character names on new partitions
8. Ability to specify current logical disk in absolute tree names (<*> disk name)
9. LISTF option to print passwords of sub-directories
10. LISTF option to print the time/date modified stamp (DTM)

2.2 NAMING CONVENTIONS

The PRIMOS file structure on any disk pack is a tree structure where the MFD is the root or trunk of the tree, the links between directories and files or subdirectories are branches, and the directories and files are nodes. A directory tree consists of all files and subdirectories that have their root in that directory. In Figure 2-1, the directory tree for UFD1 is circled. An MFD directory path name consists of a list of directories and passwords necessary to move down the tree from the MFD to any directory. For example, the MFD path name for SUFD11 is:

```
MFD MFDPASSWORD > UFD1 UFD1PASSWORD > SUFD11 UFD11PASSWORD
```

The ">" separates directories in the path-name and suggests that one is proceeding down a tree structure. Note that file and directory names can be as long as 32 characters (6 on old partitions), but passwords can be at most 6 characters long.

An MFD directory path-name may optionally omit the MFD and may optionally include the logical disk number of the pack or the pack name as shown below:

```
UFD1 UFD1PASSWORD > SUFD11 SUFD11PASSWORD
< 1 > UFD1 UFD1PASSWORD > SUFD11 SUFD11PASSWORD
< TSDISK > UFD1PASSWORD > SUFD11 SUFD11PASSWORD
```

The logical disk number may optionally follow the first ufd as follows:

```
UFD1 UFD1PASSWORD 1 > SUFD11 SUFD11PASSWORD
```

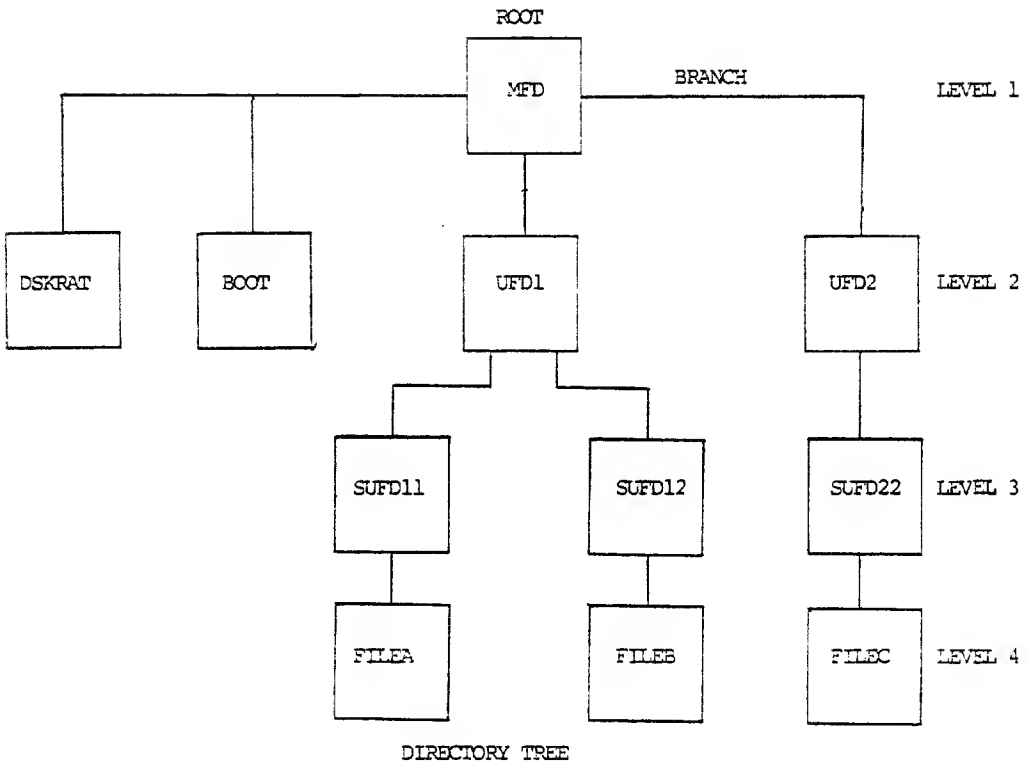
If no pack name or disk number is given, the logical disk referred to is the lowest numbered logical disk in whose MFD UFD1 appears. A user, using the ATTACH or PRIMOS LOGIN command or the FUTIL ATTACH command may specify a particular user-file-directory in the file structure as the home-ufd. Additional FUTIL ATTACH commands may refer to either the MFD or the home-ufd as the starting point. If the logical disk name is specified as "*", the MFD of the current logical disk is scanned for UFD1. Names of this form are referred to as absolute path names since the location of the home UFD is not part of the name. A home-ufd directory path name consists of a list of directories and passwords necessary to move down the tree from the home-ufd to any directory which has the home-ufd as the root. For example, if the home-ufd is UFD1, the home-ufd path name to SUFD11 would be:

```
* > SUFD11 SUFD11PASSWORD
```

"*" represents the home-ufd. The home-ufd path-name to UFD1 is simply "*". This form of tree name is also referred to as a relative path name.

Whereas many users are familiar with user-file-directories, few are familiar with a second type of directory called a segment directory. A user-file-directory is a file which consists of a header and a number of entries. Each entry consists of a 1 to 32 character filename (on old partitions, names can be at most 6 characters long), protection attributes of the file, and a disk record address pointer to the file. A segment directory is a file consisting of as many as 65535 entries, each entry being a disk record address pointer to the file. A Ø pointer indicates no file at that entry. To refer to a particular file in a segment directory, a user must specify the entry number of the entry in the segment directory. A user specifies the position as an unsigned entry number, enclosed in parentheses.

Figure 2-1. Sample File Structure



The first file is referred to as (0), the second as (1), the 440th file as (439), and the 441st file as (440). The construction (entry number) will be referred to as a segment directory filename. In FUTIL, arguments to the commands are either user-file-directory filenames or segment directory filenames depending on the directory type the file is under. Furthermore, names typed on the LISTF command of FUTIL are of either type depending on the directory type the file is under.

2.3 DESCRIPTION OF FUTIL COMMANDS

To invoke FUTIL, type FUTIL. When loaded, FUTIL types the ">" prompt character and awaits a command string from the user terminal. To terminate long operations such as LISTF, type CTRL P and restart FUTIL at 1000. A user should type a command followed by a carriage return and wait for the prompt character before typing the next command. The erase character " and the kill character ? may be used to modify the command string. In the following description of commands, underlined letters represent the abbreviation of the command or argument. [] surround optional arguments. ...means the previous element may be repeated.

* Indicates following information is a comment

QUIT return to PRIMOS Command Mode

FROM directory-path-name

where directory-path-name is of format

<LDISK> DIRECTORY [PASSWORD] [LDISK] > DIRECTORY [PASSWORD]...
or <DSKNAM>
or < * >

FROM defines the from-directory in which files are to be searched for the commands COPY, COPYSAM, COPYDAM, DELETE, LISTF, LISTSA, SCAN, CLEAN, PROTECT, TREPRO, UFDPRO, TRECPY, TREDEL, UFDCPY, and UFDDEL. The from-directory is defined from the directory-path-name whose format is given above and described in detail in Section 2.2. The path-name may contain at most 10 directories which may be segment directories as well as user-file-directories. If segment directories are specified, the user must have read access rights to them. If any error is encountered, the from-directory is set to home-ufd (*). The first directory in the path name may be "*" which refers to the home-ufd. The default from-directory is the home-ufd. Note that the FROM command never changes the home-ufd. If the FROM name is a relative path name (i.e. beginning with *>), any future ATTACH's, which do change the home-ufd, will reset the FROM name to *.

Examples:

FROM <Ø> CARLSON

Set from-directory to CARLSON on logical disk Ø. CARLSON must be in the MFD on logical disk Ø and have a blank password.

FROM CARLSON ABC

Search the MFD on all started disks for CARLSON in logical disk order Ø - 8. Set the from-directory to the first CARLSON directory found. One of the passwords of CARLSON must be ABC.

FROM <TSDISK> CARLSON > SUB1 > SUB2

Set the from-directory to SUB2. SUB2 must be a directory in SUB1; SUB1 must be a directory in CARLSON; and CARLSON must be a dirctory in the MFD on a disk with pack name TSDISK. CARLSON, SUB1, and SUB2 must have a blank password.

FROM *

Set the from-directory to the home-ufd. The home-ufd is normally the last ufd the user has logged into, or attached to with either the ATTACH or FUTIL ATTACH command. If logged into CARLSON, the above command sets the from-directory effectively to CARLSON. This command does not have to be given again if the user changes the home-ufd. Furthermore, this command does not have to be given at all unless the from-directory has been made something other than home, as home-ufd is the default.

FROM * > SUB1

Set the from-directory to SUB1. SUB1 must be a directory in the home-ufd and have a blank password.

TO directory-path-name

TO defines the to-directory in which files are searched for the commands CREATE, COPY, COPYSAM, COPYDAM, TRECPY, and UFDCPY. The to-directory is defined from the directory-path-name. The path-name may contain at most 1Ø directories which may be segment directories as well as user-file directories. If segment directories are specified, the user must have read and write access to them. The first directory in the path-name may be *. The default to-directory is the home-ufd. If any error is encountered, the to-directory is set to home-ufd (*).

Note that the TO command never changes the home-ufd. If the TO name is a relative path name (i.e. beginning with *>), any future ATTACH's, which do change the home-ufd, will reset the to-name to *.

ATTACH directory-path-name

ATTACH moves the home-ufd to the directory defined by the directory-path-name. The path name may contain at most 10 directories. The first directory in the path-name may be *. All directories in the path-name must be user-file-directories. If segment directories are specified within the path-name, a "BAD STRUCTURE" error will be reported and the home-ufd will be set to the last UFD specified in the path name before the error. An attach command will reset relative "to" and "from" path names to * but leaves absolute names alone.

COPY FILEA [FILEB] [, FILEC [FILED]]...

Copy FILEA in the from-directory to FILEB in the to-directory and optionally FILEC in the from-directory to FILED in the to-directory. If FILEB is omitted, the new file is given the same name as the old file. FILEA and FILEC must be SAM or DAM files and cannot be directories. R M or DAM files and cannot be directories. Read access rights are required for FILEA and FILEC. If FILEB exists prior to the copy, it must be a SAM or DAM file and have read, write, and delete/truncate access rights. The file type of FILEB will be made the same as FILEA.

Examples:

COPY FILEA

copy FILEA in the from-directory to FILEA in the to-directory

COPY FILEA , FILEB , FILEC

Copy FILEA, FILEB, and FILEC in the from-directory to FILEA, FILEB, and FILEC in the to-directory.

COPY FILEA FILEB

copy FILEA in from-directory to FILEB in to-directory

COPY FILEA1 FILEA2,FILEB1 FILEB2,FILEC1 FILEC2

copy FILEA1, FILEB1, and FILEC1 in the from-directory to FILEA2, FILEB2, and FILEC2 in the to-directory

COPY (Ø)

The from-directory and to-directory must each be segment directories. Copy the file at position (Ø) of the from-directory to position (Ø) of the to-directory. There are no access rights attached to these files, so PRIMOS checks instead the access rights of the directory. No spaces are allowed in the name (Ø).

COPY (Ø) (1)

Copy the file at position (Ø) of the from-directory to position (1) of the to-directory.

COPYSAM FILEA [FILEB] [, FILEC [FILED]]...

same as COPY but also set file type of FILEB and FILED to SAM instead of copying the type of FILEA and FILEC.

COPYDAM FILEA [FILEB] [, FILEC FILED]]...

same as COPYSAM but set file type of FILEB and FILED to DAM

TRECPY DIRA [DIRB] [, DIRC [DIRD]]...

Copy the directory tree specified by directory DIRA to directory DIRB and optionally DIRC to DIRD. DIRB and DIRD must not previously exist. If DIRB is omitted, use name DIRA as the directory to copy to. A directory tree consist of all files and sub-directories that have their root in that directory. DIRA and DIRC must be in the from-directory. DIRB and DIRD are created in the to-directory. Read access rights are required for DIRA and DIRC and all files or sub-directories within them. The restriction on sub-directories can be overridden with the FORCE command.

DIRB and DIRD are created with the same directory type and passwords as DIRA and DIRC and with default access rights. The names, access rights and passwords of all files and sub-directories copied are also copied.

Example:

```
FROM  MFD
TO    MFD
TRECPY   CARLSON   CARNEW
```

copy the directory tree specified by CARLSON in the MFD to a new directory CARNEW in the MFD

UFDCPY

Copy all files and directory trees from the from-directory to the to-directory. The user must have owner rights in the FROM directory. Furthermore, all files and directories in the from-directory, as well as all sub-directories and the files within them, must have read access rights. This restriction on sub-directories can be overridden with the FORCE command. Files already existing in the to-directory with names identical to those in the from-directory are replaced. Files that are replaced must have read, write, and delete accesss rights.

Segment directories already existing in the to-directory with names identical to those in the from-directory are not allowed and will not be copied. Files and directories created in the to-directory will have the same file type and access rights as the old files. If a file or UFD in the to-directory has the same name as a file or UFD in the from-directory, the access rights must permit read, write, and truncate/delete. When the copy is finished, the new file will have the same protection attributes as the corresponding file in the from-directory. The names, access rights and passwords of all files and sub-directories within directory trees being copied are also copied. Other existing files and directories in the to-directory are not affected. UFDCPY is effectively a merge of two directories including merging sub-UFD's. Both the from and the to-directory must be user-file directories.

Example:

```
FROM  CARLSON
TO    CARNEW
UFDCPY
```

copies all files and directories from CARLSON in the MFD to CARNEW in the MFD. Note that unlike the example for TRECPY, the user has not specified the MFD as the from-directory, therefore, does not need to know the MFD password. In the example CARNEW exists prior to the UFDCPY. With TRECPY, CARNEW does not previously exist.

CREATE UFDNAME [OWNERPASSWORD [NONOWNERPASSWORD]]

Creates a UFD in the "TO" directory with the owner and non-owner passwords specified. A UFD of the same name cannot already exist in the "to" directory. If a password is not specified, it will be set to 6 blanks. If a password is specified - longer than 6 characters, only the first 6 will be used. The access rights of the new UFD will be the default rights assigned by PRIMOS.

DELETE FILEA [FILEB] ...

delete FILEA and optionally FILEB from the from-directory. FILEA and FILEB cannot be directories. The user must have read, write and delete access rights to each file. If FILEA and FILEB are in a segment directory, read, write and delete access rights are required for the from-directory.

Examples:

DELETE FILEA

DELETE FILEA FILEB FILEC FILED

TREDEL DIRA [DIRB] ...

delete the directory tree specified by directory DIRA and optionally delete DIRB from the from-directory. DIRA and DIRB must be directories. The user must have read, write, and delete rights to DIRA and DIRB. Read, write and delete rights are not required for files and sub-directories nested within DIRA or DIRB. If FILEA and FILEB are in a segment directory, read, write, and delete access rights are required for the from-directory. Note that the operating system DELETE command will not delete a directory on a new partition if it is not empty.

UFDDEL

delete all files and directory trees within the from-directory. User must give the owner password in the FROM command and have read, write, and delete access to all files and directories within the from-directory. These rights are not required for files and sub-directories nested within the directories in the from-directory. Note that read and write access rights to a sub-UFD are sufficient to delete the contents of that directory, but not the directory itself.

LISTF [level] [LSTFIL] [PROTECT] [SIZE] [TYPE] [DATE] [PASSWORDS] [FIRST]

List the from directory-path-name, the to directory-path name, and all files and directory trees in the from-directory on the terminal. Optionally, follow each file by its protection attributes, size in disk records (mod 440 words), file type, date/time modified (DTM), and, on directories, the owner and non-owner passwords. The user must give the owner password in specifying the from-directory. If the LSTFIL option is given, the list of files is sent to file "LSTFIL" in the home-ufd instead of to the terminal. At a later time, a user may wish to print that file on a line printer. Level is a number specifying the lowest level in the from-directory tree structure to be listed. The following table describes the output:

level	output
0	the "from" and "to" directory names
1	the from-directory and all files and directories within it (level 1 directories)
2	all output at level 1 and all files and directories within level 1 directories
etc.	etc.

If the level is omitted, the default is 1.

The protection attribute of each file is typed as:

< owner-key non-owner-key >

The keys are number 0-7 with the following meaning:

0 no access allowed
1 read access only
2 write access only
3 read and write access
4 delete/truncate only
5 delete/truncate and read
6 delete/truncate and write
7 all access allowed

The possible file types are:

SAM	for sam file
DAM	for dam file
SEGSAM	for sam segment directory
SEGDAM	for dam segment directory
UFD	for user file directory

On new partitions, the DTM of a file or directory is printed as:

15:31:22 MON 08 NOV 1976

where 15:31:22 is 15 hours past local midnight (3PM), 31 minutes, 22 seconds. The day of the week printed will be correct for all dates between 1 January 1972 through 31 December 2071. If the date is unreasonable (e.g. when SE -0000 -0000 is typed at the system console), the DTM is not printed. All dates are considered reasonable as long as the month is between 1 and 12. Note that the day of the week will be correct for dates such as 32 December and 0 April since they will be considered as 1 January and 31 March, respectively.

The passwords on sub-directories are printed as: (OWNER, NOWNER). Note that non-printing characters are suppressed rather than replaced by blanks or printed on the user terminal although they will be sent to an output file (LSTFIL). Thus, the default passwords on a UFD are printed as (,) since the non-owner password is 0, not blanks. Similarly, a password of CNTRL (UFD) would be printed as (,) and written to a file as (^225^206^204 ,).

The FIRST line option specifies that all files not beginning with * (the usual conversion for run files) and B_ (the usual conversion for PMA and FTN object files) are to have their first lines printed. If the file is not an ASCII file and the name does not begin with * or B_, the comment "NO FIRST LINE" will be printed. First lines are preceded by ":" and will be placed on the same line as the file and its options, if it will fit. LISTF traverses the file structure as shown by the snaked line generating typeout at the various points below.

The output with level set to 3 and with the SIZE option will appear as follows for the above file structure:

```
FROM-DIR = MFD
TO-DIR   =*

BEGIN MFD              1

 DSKRAT     1 BOOT      1

 BEGIN UFD1            1
  BEGIN SUFD11          1

   FILEA       1

  END    SUFD11         2
  BEGIN  SUFD12         1

   FILEB       1

  END    SUFD12         2
 END    UFD1           5
 BEGIN  UFD2           1
  BEGIN  SUFD21         2

   FILEC       1

  END    SUFD21         2
 END    UFD2           3
END    MFD            11
```

Note that the user must have read access rights to all files, sub-directories, and files within sub-directories. This restriction can be overridden with the FORCE command.

LISTF, upon encountering a directory, prints the word BEGIN followed by the name of the directory and its size in records. On leaving a directory, LISTF prints "END Directoryname" followed by the number of records used by all files and directories within the directory tree headed by the directory file. On encountering a file, LISTF simply prints its name and size, squeezing as many files as will fit on each line. LISTF skips a line whenever a directory follows a file or a file follows a directory. LISTF will not count records in files lower than "level" in the from-directory tree. In addition, DAM file indices will not be included in the size.

In the above example, the number following MFD, 11, is the total number of records used by the MFD directory tree and consists of all files and directories on the disk pack. LISTF indents the printed output one space for each level down the tree in which the directory is located. This format makes it easy to understand the relationship of each directory to other directories in the tree.

LISTSAVE fname [level] [PROTECT] [SIZE] [TYPE] [DATE] [PASSWORDS] [FIRST]

This command is identical to LISTF with the LSTFIL option specified except the output file will be named "fname" rather than "LSTFIL" and the LSTFIL option is redundant.

SCAN fname [level] [PROTECT] [SIZE] [TYPE] [DATE] [PASSWORDS] [LSTFIL] [FIRST]

This command is used to search the from-directory tree for the occurance of all files, sub-UFD's, and segment directories named "fname". If level is specified as 1 (the default), only the file name will be printed, followed by its options. If the level is greater than 1, the path name to the file or directory, starting from the from-directory, is printed, followed by the file name and its options. For example, with the tree-structure shown for the LISTF example, the command SCAN FILEB S F 10 will print:

```
FROM=MFD
TO  =*

DIRECTORY PATH = MFD> UFD1> SUFD12

   FILEB                1 : NO FIRST LINE
```

FILEB lacks a first line since it was empty. Note that the name FILEB is indented 3 spaces since it is in a third level UFD.

CLEAN prefix [level]

This command is a conditional delete based upon a prefix match. If a file name begins with the characters specified as "prefix", the file will be deleted. If level is specified greater than 1, that many levels of sub-UFD's will be scanned for prefix matches. In no case, will CLEAN delete a UFD or a segment directory. In the example tree structure used for LISTF and SCAN, the command: CLEAN F will not delete anything since no files beginning with F exist in MFD. However, the command: CLEAN F 10 will delete ,n-300 FILEA, FILEB, and FILEC since they all begin with F. Note that: CLEAN U will not delete either

UFD1, UFD2, or any of the files within them. A typical usage of CLEAN would be:

```
CLEAN L
CLEAN B
```

To delete binary and listing files from a UFD.

PROTECT fname [owner [non-owner]]

Will protect "fname" in the from-directory with the owner and non-owner protection attributes [defined under LISTF] specified. If the non-owner rights are omitted, they will be set to 0. If the owner rights are omitted, they will be set to 1 (read only). "Fname" can be a file, a UFD, or a segment directory. If it is a UFD, the file and sub-directories within it will not be protected.

TREPRO tree-name [owner [non-owner]]

This command is essentially the same as protect except "treename" is a UFD or segment directory in the from-directory and it and all files under it (UFD's only) will be protected with the specified rights. Again, the default rights are <1 0>.

UFDPRO [owner [non-owner [levels]]]

This command is used to protect all files and directories within the from-directory with the specified rights, going down sub-UFD trees the specified number of levels. The default rights are <1 0> and the default level is 1. Thus, in the example structure of LISTF, SCAN, and CLEAN, the command: UFDPRO will protect the files DSKRAT and BOOT and the UFD's UFD1 and UFD2 with access rights <1 0> and will not change the rights of any of the sub-directory UFD's or files. The command: UFDPRO 1 0 10 will protect all files and directories within MFD. Note that both the owner and non-owner rights must be specified in order to specify the number of levels.

FORCE ON
or OFF

As noted previously, LISTF, LISTSAVE, SCAN, UFDCPY, and TRECPY will not force read access rights on any files or sub-directories within the from-directory. This is to prevent the updating of DTM's of copied files as well as permitting these commands to operated on write protected disks. The price of this

capability is that all files to be listed or copied must have read access. To override this restriction, the command FORCE ON must be specified. This will cause read access rights to be forced, but will also cause LISTF, LISTSAVE, SCAN, UFDCPY, and TRECPY to fail on write-protected disks. The option remains in force until the command: FORCE OFF is specified. Note that UFDCPY will never force rights on the primary level of either the from or to-directory.

2.4 RESTRICTIONS

FUTIL cannot process user-file-directory filenames that contain the characters "(" , ")" , "<" , ">" , "[" , "]" , or ",". Avoid using filenames containing these characters.

In using FUTIL under PRIMOS, certain operations may interfere with the work of other users. For example, a UFDCPY command to copy all files from a ufd currently used by another logged-in user may fail. If any file in that directory is open for writing by that user, UFDCPY will encounter the error FILE ALREADY OPEN, and will skip the file. If the user attempts to open one of his files for writing while UFDCPY is running, the user may encounter that errror. The FUTIL LISTF and TRECPY commands cause the same interaction problems. Other FUTIL commands such as COPY and DELETE can also interfere with the other user, but the problem is not as serious as only one file is potentially involved in a conflict. To prevent the conflicts, users working together and involved in operations using each other's directory should coordinate their activities. If two users consistently use the same ufd at the same time, they should avoid the FUTIL LISTF command, and use the system LISTF command instead.

FUTIL operations when using the MFD should be done carefully. Never give the command TREDEL MFD as the command will delete every file on the disk except the MFD, DSKRAT, BOOT, and BADSPT. A LISTF or UFDCPY of the MFD should be done only if one is sure no other user is using any files or directories on that disk. A UFDCPY of the MFD to the MFD of another disk has the effect of merging the contents of two disks onto one disk. A user should be sure there is enough room on the to-disk before attempting this operation or it will abort. Recall also that the names of segment directories on the two disks may not conflict. Files of the same name will be overwritten and UFD's of the same name will be merged. To avoid the name conflict, it may be desirable to UFDCPY the MFD of one disk into a user-file-directory on another disk. Each directory originally on the from-disk becomes a subdirectory in that ufd on the to-disk. For example, the contents of 10 diskettes could be copied into 10 user-file-directories on a 1.5 M disk pack. Note that a UFDCPY of an MFD does not copy the DSKRAT, MFD, BOOT or BADSPT to the to-directory. If a user wishes to copy BOOT to the to-directory, use the COPY command. Never copy the DSKRAT or the

BADSPT file from one MFD to another.

The effect of a UFDCPY from the MFD of a disk in use to the MFD of a newly MAKE'd disk is to reorganize the disk files so that all files are compacted, that is, have their records close to each other on the new disk. After such a compaction, the access time to existing files on the new disk is effectively reduced from the access time on the old disk. Furthermore, new files tend to be compact since all free disk records are also compacted. The use of such compacted disks should improve the performance of all PRIMOS systems.

Users should not abort copy or delete operations under DOS, but should allow them to run to completion. Aborting a copy or delete operation may cause a pointer mismatch or bad file structure or a directory with a partial entry. DOS or PRIMOS will not run correctly with a directory with a partial entry. FIXRAT should be run immediately if these conditions are encountered. Under PRIMOS III and PRIMOS IV, interruption of FUTIL with Control-P will never result in a bad file structure.

2.5 ERROR MESSAGES

The following are error messages generated by FUTIL. In many cases, FUTIL types error messages generated by DOS or PRIMOS and retains control, so users should be generally familiar with operating system error messages. The list given here includes those messages that may be encountered by FUTIL. Most messages are preceded by a file name identifying the file causing the error. Some of the error messages have the format:

```
reason for error
FILE = filename
DIRECTORY PATH = directory-path-name
```

In all cases except "DISK FULL" on copies, FUTIL will continue with the operation, reporting all errors as it goes until the operation is complete.

?

Unrecognizable command

ALREADY EXISTS or SEG DIR ALREADY EXISTS

An attempt has been made to TRECPY to or CREATE a UFD or segment directory that already exists. Or UFDCPY has attempted to copy a segment directory which already exists. If you intend to do the operation, the UFD or segment directory in the to-directory must first be deleted.

ALREADY OPEN

Indicates an attempt to UFDCPY a directory to itself or an attempt to copy a file to itself, or an attempt to copy a directory to a subdirectory within itself.

BAD NAME

A segment directory filename was given to a command which expected a ufd filename or vice versa. The type of filename must match the type of directory the file is contained in.

BAD PASSWORD

An incorrect password has been given in a FROM, TO, or ATTACH command. PRIMOS will not allow FUTIL to maintain control in case of a bad password so the FUTIL command must be given to restart FUTIL after the user has attached to his directory. The from-directory and to-directory are reset to home-ufd in this case.

BAD SYNTAX

The command line processed by FUTIL is incorrect.

CANNOT ATTACH TO SEGDIR

The last directory in the directory path name to an ATTACH command is a segment directory. It must be a ufd, as ATTACH sets the home-ufd to the last directory in the path.

CANNOT DELETE MFD

User has given the UFDDEL command while attached to the MFD. This is not allowed.

STRUCTURE TOO DEEP

Directories may be nested to a depth of 100 levels. User has attempted to exceed this limit. Under 32K PRIMOS II, this limit is dynamically reduced to 13 levels.

DISK ERROR

Same as unrecovered error.

DISK FULL

The disk has become full before FUTIL has finished a copy operation. For operations involving many files, some files are not copied, creating only partially copied directories which may be of limited use. It is suggested that the user delete such a structure immediately to prevent confusion as to what has been copied.

IN USE

Indicates a FUTIL attempt to process a file in use by some other user. It may also indicate an attempt to copy a directory to a subdirectory within itself.

BAD STRUCTURE

Indicates any of various conditions in which the implied or explicitly specified structure is illegal. For example, an attempt to specify a UFD under a segment directory will cause this error.

CANNOT COPY FILE TO DIRECTORY

On UFDCPY, indicates a file on the "from" side has the same name as a directory on the "to" side.

NO RIGHT

User has attempted an operation on a file which violates the file access rights assigned to that file. These rights may be changed by the PROTECT command, if the user has given the owner password on ATTACH.

NO UFD ATTACHED

Self-explanatory.

NOT A DIRECTORY

User has given a directory-path-name which includes a regular file.

NOT FOUND

Self-explanatory

POINTER MISMATCH

Indicates a bad file structure. Running FIXRAT is in order.

END OF FILE

User has attempted to reference a nonexistent file beyond the end of a segment directory.

NOT FOUND IN SEG-DIR

User has attempted to reference a file in a segment directory with an entry of 0, which indicates file does not exist or the user has attempted to reference a file past the end of the segment directory.

UFD FULL

On a UFDCPY merge or a UFDCPY or TRECPY from a new partition to an old partition, the to-directory or a sub-directory has become full. FUTIL will report the error and then pop-up a level and continue as if the UFD had not become full.

UNRECOVERED ERROR

Indicates either the user has attempted to write to a write-protected disk or an actual disk error or a FUTIL attempt to process a bad file structure. Running FIXRAT is in order if the disk was not write-protected.

TOO MANY NAMES

A "from", "to" or "attach" tree name was specified with more than 10 names.

WRONG FILE TYPE

An attempt was made to DELETE or copy a directory or TREDEL, TRECPY or TREPRO a file.

CANNOT ATTACH

An attempt is made to UFDCPY a directory in which a sub-ufd has the same name as a file or segment directory on the "to" side. The "from" side UFD is skipped.

PRIME

PRIME Computer, Inc., 145 Pennsylvania Avenue, Framingham, Massachusetts 01701